Java V2X SDK

User Guide

Version Control

Version	Date	Comments
3.1.0	2025/05/15	First Release

Table of Contents

1. Acronyms

Acronym	Definition
C-ITS	Cooperative ITS
C-V2X	Cellular Vehicle-to-Everything
CAM	Cooperative Awareness Message
CPM	Collective Perception Message
DENM	Decentralized Environmental Notification Message
ETSI	European Telecommunications Standards Institute
GN	Geo Networking
IVIM	Infrastructure to Vehicle Information Message
ITS	Intelligent Transport Systems
MAPEM	MAP (topology) Extended Message
MQTT	Message Queuing Telemetry Transport
POIM	Point Of Interest Message
RSU	RoadSide Unit
SDK	Software Development Kit
SPATEM	Signal Phase And Timing Extended Message
STEP	Safer Transport for Europe Platform
V2X	Vehicle-to-Everything
VAM	VRU Awareness Message
VF	VodaFone
VRU	Vulnerable Road User

2. Introduction

The **Vodafone V2X SDK** is an advanced toolkit designed to facilitate the development of V2X applications across various platforms with ease. The SDK streamlines the integration of V2X applications with the **STEP platform**, managed by Vodafone.

By managing complex processes such as connectivity with STEP, message encoding, message decoding, transmission, and reception, the SDK enables developers to concentrate solely on their business logic and user experience.

Compliant with **ETSI standards**, the combination of the **Vodafone V2X SDK** and the **STEP platform** provides a comprehensive suite of **V2X services**. These services can be seamlessly integrated into applications with minimal code requirements.

Integrating the V2X SDK allows applications to **exchange information with vehicles, pedestrians, infrastructure, and road operators** across Europe, fully leveraging the potential of **C-V2X technology** for enhanced safety and efficiency in mobility solutions.

3. STEP

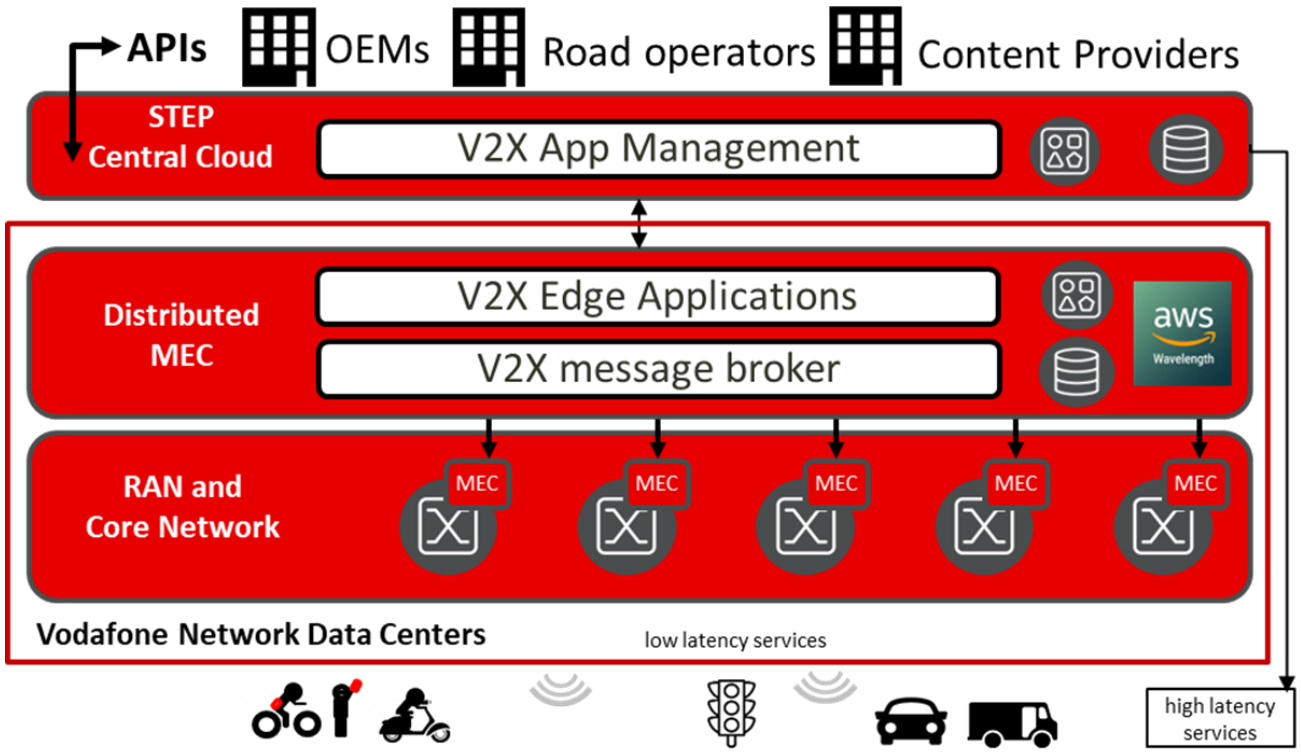
The **Safer Transport for Europe Platform** (**STEP**), managed by **Vodafone**, is an infrastructure designed to improve road safety and transportation efficiency in Europe.

A major challenge in intelligent transportation systems (ITS) is data fragmentation, where important traffic and mobility data remain isolated. The STEP platform addresses this issue by distributing data across varied traffic and transportation ecosystems, promoting interoperability between vehicles, pedestrians, road infrastructure, and transportation authorities.

The STEP platform employs advanced technologies to provide high-quality, reliable, and secure V2X services:

- **5G Connectivity** – Facilitates ultra-fast, low-latency communication for real-time data exchange.
- **Edge Computing** – Reduces response times by processing data closer to end-users and improves scalability.
- **Artificial Intelligence** (AI) – Supports decision-making, predictive analysis, and traffic optimization.

By integrating with the **Vodafone V2X SDK**, your application can connect to **STEP**, utilizing connected mobility and road safety services across Europe.



4. ETSI

The **European Telecommunications Standards Institute** (ETSI) is an independent organization responsible for developing globally accepted standards in telecommunications.

ETSI plays a crucial role in defining standards for **Vehicle-to-Everything (V2X) communication**, ensuring consistency and interoperability across different implementations.

The **Vodafone V2X SDK** fully adheres to **ETSI standards**, guaranteeing compatibility with third-party **V2X applications**, whether they are built using alternative V2X technology stacks or other V2X SDKs.

This compliance enables seamless data exchange and interoperability, fostering a unified and standardized C-V2X ecosystem.

5. Messages Supported by the V2X SDK

The **Vodafone V2X SDK** supports the exchange of multiple **Intelligent Transport Systems (ITS)** messages with the **STEP** platform.

All supported message types (Note1) conform to **ETSI standards**, ensuring interoperability within the broader V2X ecosystem.

Note1: Except the Custom Message.

The SDK can operate in different roles depending on the message type:

- **Producer**: The SDK **encodes and publishes** messages to the STEP platform.
- **Consumer**: The SDK **subscribes to and decodes** incoming messages from the STEP platform.
- **Bidirectional**: In some cases, the SDK can act as **both a producer and a consumer**, handling message transmission and reception simultaneously.

By leveraging these capabilities, developers can seamlessly integrate **real-time V2X communication** into their applications.

Message Type	Producer	Consumer
CAM	x	x
VAM	x	x
DENM	x	x
IVIM		x
SPATEM		x
MAPEM		x
CPM		x
POIM		x
Custom Message (not standardized by ETSI)	x	x

6. Standards Supported by the V2X SDK

6.1. C-ITS Messages

C-ITS Message	ETSI Standard	ASN files
CAM (Release2)	ETSI TS 103 900 V2.1.1 (2023-11)	CAM-PDU-Descriptions {itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) camPduRelease2 (103900) major-version-2 (2) minor-version-1 (1)} # Dependencies: ETSI-ITS-CDD {itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) 102894 cdd (2) major-version-4 (4) minor-version-1 (1)}
DENM (Release2)	ETSI TS 103 831 V2.2.1 (2024-04)	DENM-PDU-Description {itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) denmPduRelease2 (103831) major-version-2 (2) minor-version-2 (2)} # Dependencies: ETSI-ITS-CDD {itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) ts (102894) cdd (2) major-version-4 (4) minor-version-1 (1)}
ITS-Container / Common Data Dictionary	ETSI TS 102 894-2 V2.3.1 (2024-08)	ETSI-ITS-CDD {itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) 102894 cdd (2) major-version-4 (4) minor-version-2 (2)}
IVIM	ETSI TS 103 301 V2.2.1 (2024-08)	IVIM-PDU-Descriptions { itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) ts103301 (103301) ivim (2) major-version-2 (2) minor-version-1 (1)} # Dependencies: ETSI-ITS-CDD { itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) 102894 cdd (2) major-version-4 (4) minor-version-1 (1)} IVI { iso (1) standard (0) ivi (19321) major-version-3 (3) minor-version-1 (1) } EfcDataDictionary {iso(1) standard(0) 17573 dd(3) version1(1) minorVersion2(2)} GddVersion1 {iso(1) standard(0) gdd(14823) part1(1) gdd-version-1(1) revision-1(1)} IVI-IS
SPATEM	ETSI TS 103 301 V2.2.1 (2024-08)	SPATEM-PDU-Descriptions { itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) ts103301 (103301) spatem (0) major-version-2 (2) minor-version-1 (1)} # Dependencies ETSI-ITS-CDD { itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) 102894 cdd (2) major-version-4 (4) minor-version-1 (1)} ETSI-ITS-DSRC {itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) ts103301 (103301) dsrc (6) major-version-2 (2) minor-version-1 (1)} ETSI-ITS-DSRC-REGION {itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) ts103301 (103301) dsrc (6) region (1) major-version-2 (2) minor-version-1 (1)} ETSI-ITS-DSRC-AddGrpC {itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) ts103301 (103301) dsrc (6) addgrpc (0) major-version-2 (2) minor-version-1 (1)}

MAPEM	**ETSI TS 103 301 V2.2.1 (2024-08)**	MAPEM-PDU-Descriptions {itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) ts103301 (103301) mapem (1) version2 (2)} # Dependencies ETSI-ITS-DSRC { itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) ts103301 (103301) dsrc (6) major-version-2 (2) minor-version-1 (1)} ETSI-ITS-CDD { itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) 102894 cdd (2) major-version-4 (4) minor-version-1 (1)}
VAM	**ETSI TS 103 300-3 V2.2.1 (2023-02)**	VAM-PDU-Descriptions {itu-t(0) identified-organization(4) etsi(0) itsDomain(5) wg1(1) 103300 vam(1) major-version-3(3) minor-version-1(1)} #Dependencies ETSI-ITS-CDD {itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) 102894 cdd (2) major-version-3 (3) minor-version-1 (1) } ETSI-ITS-CDD {itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) 102894 cdd (2) major-version-4 (4) minor-version-1 (1)}
CPM	**ETSI TS 103 324 V2.1.1 (2023-06)**	CPM-PDU-Descriptions { itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) ts (103324) cpm (1) major-version-1 (1) minor-version-1(1)} # Dependencies ETSI-ITS-CDD {itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) ts (102894) cdd (2) major-version-3 (3) minor-version-1 (1)} ETSI-ITS-CDD {itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) 102894 cdd (2) major-version-4 (4) minor-version-1 (1)} CPM-OriginatingStationContainers {itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) ts (103324) originatingStationContainers (2) major-version-1 (1) minor-version-1(1)} CPM-SensorInformationContainer {itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) ts (103324) sensorInformationContainer (3) major-version-1 (1) minor-version-1(1)} CPM-PerceptionRegionContainer {itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) ts (103324) perceptionRegionContainer (5) major-version-1 (1) minor-version-1(1)} CPM-PerceivedObjectContainer {itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) ts (103324) perceivedObjectContainer (4) major-version-1 (1) minor-version-1(1)}
POIM	**ETSI TS 103 916 v2.1.1 (2024-01)**	POIM-PDU-Description {itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) poi(103916) poim(0) major-version-1 (1) minor-version-1 (1)} # Dependencies ETSI-ITS-CDD {itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) ts (102894) cdd (2) major-version-4 (4) minor-version-1 (1)} WITH SUCCESSORS POIM-ParkingAvailability {itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) poi(103916) parkingAvailability(1) major-version-1 (1) minor-version-1 (1)} WITH SUCCESSORS POIM-CommonContainers {itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) poi(103916) common(99) major-version-1 (1) minor-version-1 (1)} EfcDataDictionary {iso(1) standard(0) 17573 dd(3) version1(1) minorVersion3(3)}

6.3. Headers

Header Name	Standard
GeoNetworking Header	ETSI EN 302 636-4-1 V1.4.1 (2020-01)
Basic Transport Protocol Header	ETSI EN 302 636-5-1 V2.2.1 (2019-05)

7. Overview of C-ITS Messages

This chapter explains the Cooperative Intelligent Transport Systems (C-ITS) messages supported by the SDK, including their purpose, structure, and role within the V2X ecosystem.

7.1. Cooperative Awareness Messages (CAM)

Cooperative Awareness Message (CAM) is a crucial part of Intelligent Transport Systems (ITS) for real-time situational awareness. Periodically broadcast by ITS stations, such as vehicles, cyclists, and pedestrians, CAMs share sender status, position, and dynamics to improve road safety and traffic efficiency.

A CAM includes an ITS PDU header and the following containers:

- **Basic container**: ITS-S type and geographic position.
- **High-frequency container**: Fast-changing data (e.g., speed, heading).
- **Low-frequency container**: Less dynamic data (e.g., light status).
- **Special vehicle container**: Additional data for emergency, public transport, or road work vehicles.

7.2. Decentralized Environmental Notification Messages (DENM)

Decentralized Environmental Notification Message (DENM) is used in Intelligent Transport Systems (ITS) to convey **road hazards** and **abnormal traffic conditions**.

A DENM includes a common ITS PDU header and the following containers :

- **Management container** – Contains event metadata like action ID, detection time, reference time, event position, validity duration, and station type.
- **Situation container** – Specifies the event type and might include linked causes, event zones, related DENMs, and event termination data.
- **Location container** – Details event location including speed, road type, lane positions, and detection zones.
- **À La Carte container** – Holds optional application-specific data.

7.3. Infrastructure to Vehicle Information Messages (IVIM)

The **Infrastructure to Vehicle Information Message** (IVIM) is a C-ITS message used to broadcast road infrastructure information from roadside units (RSUs) or other infrastructure elements to vehicles. It enhances situational awareness by providing static and dynamic road data, thus improving traffic efficiency and safety.

An IVIM includes a common ITS PDU header and the following containers:

- **Management container**: Includes ServiceProvider ID, status, and validity dates.
- **Geographic Location container**: Provides Reference Position and GlcParts, specifying geographic location details such as zoneID and optional laneNumber.
- **General IVI container**: Contains GicParts with information like Vienna Convention road signs, ISO/TS 14823 pictograms, and ITIS codes.

7.4. Signal Phase and Timing Extended Messages (SPATEM)

The SPATEM (Signal Phase and Timing) message delivers **real-time data** on the **phases and timing of traffic signals at intersections**. It comprises a common ITS PDU header and a SPAT container. The SPAT container includes a **timestamp**, an **optional descriptive name**, an **IntersectionStateList**, and an **optional regional extension**.

Important: The SPATEM message provides the timing details for each traffic light at an intersection but does not include their geographical position or the lanes involved. This information must be obtained from the associated MAPEM message.

7.5. MAP (Topology) Extended Messages (MAPEM)

MAP Extended Message (MAPEM) describes an intersection's topology, including:

- Lane direction
- Lane type
- Lane coordinates
- Traffic Light ID linked to the lane
- Maneuvering
- Maximum speed
- Altitude
- Additional details

7.6. VRU Awareness Messages (VAM)

The **VRU Awareness Message** (VAM) is a C-ITS message aimed at enhancing the safety of Vulnerable Road Users (VRUs) like pedestrians, cyclists, and motorcyclists. It allows vehicles and infrastructure to detect and track nearby

VRUs by sharing their position, movement, and status. VAM messages improve situational awareness and help prevent collisions by making VRUs visible to connected vehicles and road infrastructure in real-time.

A VAM includes an ITS PDU header and several containers:

- **Basic container** – ITS-S type and geographic position.
- **High-frequency container** – Fast-changing data (e.g., speed, heading, longitudinal acceleration).
- **Low-frequency container** – Less dynamic information (profile, sub profile, exterior lights).
- **Motion Prediction container** – Path history and prediction.
- **Cluster Operation container**
-

7.7. Collective Perception Messages (CPM)

The Collective Perception Message (CPM) is utilized by an Intelligent Transport System station (ITS-S) to broadcast data regarding **detected objects**, such as **vehicles, pedestrians, and animals**, as well as **perception regions**. By disseminating information about objects and surrounding areas, the CPM enhances ITS-S's **environmental awareness**. This is achieved through the sharing of data about detected objects in the environment, collected by vehicle and infrastructure sensors.

The CPM consists of a common ITS PDU header and the CPM payload. The CPM payload contains the following containers:

- **Management Container**
- **Originating Vehicle Container**
- **Originating RSU Container**
- **Sensor Information Container**
- **Perception Region Container**
- **Perceived Object Container**

7.8. Point of Interest Messages (POIM)

The **Point of Interest Message (POIM)** is used to broadcast information about points of interest, particularly parking spaces.

It allows an Intelligent Transport System (ITS) to share data on parking availability, status, and associated services.

The POIM-PA consists of both generic and parking-specific information, which is periodically transmitted or disseminated based on predefined rules.

This message is a key component of the Parking Availability Service (PAS), enhancing real-time parking management and user accessibility.

The POIM includes a common ITS PDU header and up to 8 Parking Availability blocks.

Each Parking Availability block includes the following containers:

- **Management Container** (Service provider , timestamp ...)
- **Place Info Container** (location , address , opening hours ...)
- **Aggregated Status Container** (status, totalSpaces , freeSpaces ...)

8. Building Your First V2X Application

This chapter guides you through the process of building your first V2X application using the Vodafone V2X SDK. It covers the prerequisites, dependencies, and essential configurations required to integrate the SDK into your project.

You will learn how to:

- Set up and configure the V2X SDK
- Start and stop the V2X Main Service
- Subscribe to and unsubscribe from events
- Use various V2X sub-services, including CAM, DENM, IVIM, SPATEM, MAPEM, VAM, CPM, and POIM
- Send and receive custom messages

By the end of this chapter, you will have a fully functional V2X application, capable of exchanging messages with the STEP platform and interacting with other V2X-enabled systems.

8.1. Prerequisites

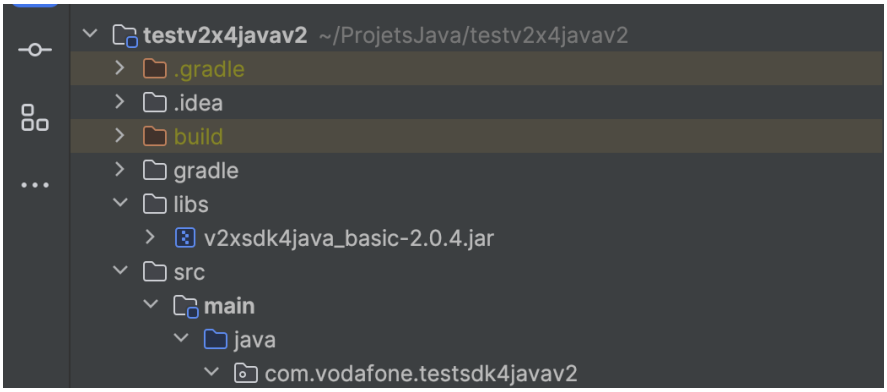
Before you begin, ensure that you have the following installed and set up on your system:

- IntelliJ IDEA Community Edition installed on your laptop.
- The latest version of the Java V2X SDK library and its corresponding Javadoc.
- Your application is registered on the STEP web portal, and you have received your credentials (ApplicationID and ApplicationToken).

Note: If you do not have the required SDK files, you can download them from the STEP web portal.

8.2. Dependencies

Copy the JAVA V2X SDK library in the libs folder of your project



Add the dependency in your build.gradle

```
dependencies {
    testImplementation platform('org.junit:junit-bom:5.9.1')
    testImplementation 'org.junit.jupiter:junit-jupiter'
    // V2XSDK4JAVAV2 Library
    implementation files('libs/v2xsdk4java_basic-2.0.4.jar')
    // Log4J Libraries
    // https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-api
    implementation group: 'org.apache.logging.log4j', name: 'log4j-api', version: '2.19.0'
    // https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-core
    implementation group: 'org.apache.logging.log4j', name: 'log4j-core', version: '2.19.0'

}
```

8.3. V2X SDK

8.3.1. Configure V2X SDK

Before using the SDK, you must configure it.

- The configuration is defined at startup and **cannot be modified during execution**.
- Configuration is done using a ***SDKConfiguration*** object.
- At a minimum, you need to provide your **credentials.**

IMPORTANT: Without valid credentials your app will not be able to connect to the STEP servers.

Example:

```
// SDKConfiguration

SDKConfiguration config = SDKConfiguration.builder()
.stationType(StationType.PASSENGER_CAR).vehicleRole(VehicleRole.DEFAULT)
.camServiceMode(ServiceMode.TxAndRx).camPublishGroup("public").camSubscribeGroup("public")
.camZonesAlgorithmFactory(new SDKZoneAlgorithmFactory(SDKZoneAlgorithmFactory.SDKZoneAlgorithm.Basic))
.applicationID(MY_APP_ID).applicationToken(MY_APP_TOKEN).build();
```

8.3.2. Provide a Location Provider

The **V2X SDK requires access to the user's current location**.

To achieve this, you must create a Class implementing the **LocationProvider** interface.

This class oversees retrieving and supply's location data to the SDK.

For testing purposes, you can use the **FakeGNSSReceiver** class provided in the SDK.

Example:

```
// A fake LocationProvider

double refLat=48.866667d;
double refLon=2.333333d;
final GnssLocation lastLocation ;

LocationProvider locationProvider = new FakeGNSSReceiver(refLat,refLon);
```

This example initializes a **fake location provider** with predefined latitude and longitude values, allowing you to simulate static GNSS data for testing your application.

Note: The source code of this class is provided in the Annexes.

8.3.3. Create an Instance of V2XSDK

You just need to pass to the constructor the LocationProvider & the Configuration object.

Note: Your application can create multiple instances of the V2X SDK.

Example:

```
// Create an instance of the V2XSDK

V2XSDK sdk = new V2XSDK(locationProvider,config);
```

8.3.4. Start the V2X Service

Example:

```
// Start the V2X SDK

 sdk.startV2XService();
```

8.3.5. Wait for V2X Service Initialization

Before using the V2X SDK, you must ensure that the **V2X Service** is fully initialized. To do this, monitor the **V2XService state** and wait until it switches to **UP_AND_RUNNING**.

Example:

```
// Wait the completion of the V2XService Initialization

while (sdk.getV2XServiceState() != V2XServiceState.UP_AND_RUNNING){
    Thread.sleep(1000L);
 }
```

Note: Subscribe to the V2X_SERVICE_STATE_CHANGED event to get notified of any V2XService state changes.

8.3.6. Wait for Connectivity with STEP is Established

Once the V2X Service initialization is completed, it is highly recommended to check the **connectivity with the STEP servers.**

Example:

```
// Check the V2XService is connected to STEP

while (sdk.getV2XConnectivityState() != V2XConnectivityState.CONNECTED){
  Thread.sleep(1000L);
}
```

Note: Subscribe to the V2X_CONNECTIVITY_STATE_CHANGED *event to get notified of any V2X Connectivity state changes.*

8.3.7. Subscribing to SDK Events

Subscribing to SDK Events

To receive SDK events in your application, you need to:

1. Implement an EventListener callback to handle incoming events.
2. Subscribe to the specific SDK events you want to receive.

Example:

```
EventListener eventListener = new EventListener() {

    @Override
     public void onMessageBusEvent(BaseEvent baseEvent) {
        logger.info("Incoming SDK Event , "+baseEvent.getEventName()+" => "+gson.toJson(baseEvent));
    }
};

 sdk.subscribe(eventListener, EventType.V2X_CONNECTIVITY_STATE_CHANGED,
EventType.V2X_SERVICE_STATE_CHANGED,
EventType.APPLICATIVE_SERVICE_STATED_CHANGED,
EventType.FRESH_GPS_NAV,
EventType.CAM_LIST_CHANGED,
EventType.DENM_LIST_CHANGED);
```

8.3.8. CAM Service

Configuration

The CAM Service automatically handles the transmission and reception of Cooperative Awareness Messages (CAMs).

- By default, the service operates in TxAndRx mode, using the "public" group for both publishing and subscribing.
- If you created a private application, you must update the camSubscribeGroup and camPublishGroup parameters accordingly.
- To receive only CAM messages from nearby vehicles, configure the CAM service in RxOnly mode.
- To transmit CAM messages without subscribing to others, configure the CAM service in TxOnly mode.

To modify the CAM Service configuration, use the SDKConfiguration object. For more details, refer to the chapter "Configuring the SDK"

Starting & Stopping the CAM Service

The CAM Service can be started or stopped at any time, provided that the V2X Service is already running. If the V2X Service has not been started, attempting to start the CAM Service will result in an IllegalStateException.

Example:

```
// Start the CAM Service
 sdk.startCAMService();
 // Stop the CAM Service *
sdk.stopCAMService();
```

Handling Incoming CAM

To handle CAM emitted by nearby vehicles your application needs to subscribe to the Event **CAM_LIST_CHANGED**

Example:

```
EventListener camListener = new EventListener() {
 @Override
 public void onMessageBusEvent(BaseEvent baseEvent) {
  if (baseEvent.getEventType()==EventType.CAM_LIST_CHANGED){
   EventCamListChanged ecl = (EventCamListChanged) baseEvent;
  // process the incoming CAMs here
   List<CAMRecord> camRecords = ecl.getList();
    for(CAMRecord camRecord : camRecords){
      // Implement your logic
      long stationID = camRecord.getStationID();
      ...
    }
  }
 }
 };
sdk.subscribe(camListener, EventType.CAM_LIST_CHANGED);
```

8.3.9. DENM Service

Configuration

The DENM Service automatically handles the transmission and reception of DENMs.

- By default, the service operates in TxAndRx mode, using the "public" group for both publishing and subscribing.
- If you have created a private application, you must update the denmSubscribeGroup and denmPublishGroup parameters accordingly.
- To receive only DENM messages from nearby vehicles, configure the service in RxOnly mode.
- To transmit DENM messages without subscribing to others, set the service to TxOnly mode.

To modify the DENM Service configuration, use the SDKConfiguration object. For more details, refer to the chapter "Configuring the SDK"

Starting & Stopping the DENMM Service

The DENM Service can be started or stopped at any time, provided that the V2X Service is already running. If the V2X Service has not been started, attempting to start the DENM Service will result in an IllegalStateException.

Example:

```
// Start the DENM Service
sdk.startDENMService();

// Stop the DENM Service
sdk.stopDENMService();
```

Handling Incoming DENMs

To handle DENM emitted by nearby vehicles your application needs to subscribe to the Event DENM_LIST_CHANGED

Example:

```
EventListener denmListener = new EventListener() {
    @Override
    public void onMessageBusEvent(BaseEvent baseEvent) {
      if (baseEvent.getEventType()==EventType.DENM_LIST_CHANGED){
          EventDenmListChanged edl = (EventDenmListChanged) baseEvent;
          // process the incoming DENMs here
          List<DENMRecord> denmRecords = edl.getList();
          for(DENMRecord denmRecord : denmRecords){
             // Implement your logic long
             originatorStationID = denmRecord.getOriginatorID();
             int causeCode = denmRecord.getCauseCode();
              .....
          }
      }
    }
 };
 sdk.subscribe(camListener, EventType.DENM_LIST_CHANGED);
```

Trigger a DENM

To trigger a **DENM**, call the **denmTrigger()** method and pass the **DENMType** and the **location** as arguments.

```
/**
 * Triggers a predefined DENM.
 *
 * @param denmType                  The DENM to be triggered
 * @param location                  The DENM location
 * @return                          The sequence Number of the DENM
 * @throws IllegalStateException    Exception if the V2XService or the DENM Service is not running
 */
```

Example:

```
GnssLocation accidentLocation = new GnssLocation(48.866667d,2.3333333d,null,null,null,null,sdk.getUTCTimeInMs());

long sequenceNumber = sdk.denmTrigger(DENMType.ACCIDENT, accidentLocation);
```

Update a DENM

To update a **DENM**, call the **denmUpdate()** method, providing the **sequenceNumber** (mandatory) and the parameters you want to modify. Set **null** for any parameters that should remain unchanged.

```
/**
 * Updates an existing DENM
 * Note: Set to null the fields to let unchanged.
 *
 * @param sequenceNumber            The sequence number of the DENM to be updated (mandatory)
 * @param termination               The Termination parameter (optional)
 * @param causeCode                 The Cause Code (optional)
 * @param subCauseCode              The Sub Cause Code (optional)
 * @param refLocation               The reference location (optional)
 * @param durationInSecond          The duration in seconds (optional)
 * @param intervalInMs              The repetition interval (ms)
 * @param relevanceDistanceInMeter  The Relevance Distance (meter)
 * @return                          True , if the DENM specified using the sequenceNumber exists.
 * @throws IllegalStateException    Exception if the V2XService or the DENM Service is not running
 */
```

Example:

```
// Change the durationInSecond to 100s & the repetionIntervalInMs to 500ms
sdk.denmUpdate(sequenceNumber,null,null,null,null,100,500,null);
```

Terminate a DENM

To terminate a **DENM**, call the **denmTerminate()** method and pass the **sequenceNumber** returned by the SDK when the **DENM** was triggered.

```
/**
 * Terminates a DENM.
 *
 * @param sequenceNumber                The sequence Number of the DENM to be terminated
 * @return                              True, if the DENM specified using the sequenceNumber exists
 * @throws IllegalStateException        Exception if the V2XService or the DENM Service is not running
 */
```

Example:

```
sdk.denmTerminate(sequenceNumber);
```

8.3.10. IVIM Service

Configuration

The IVIM Service handles the reception of Infrastructure to Vehicle Information Message (IVIMs).

- By default, it uses "public" group for subscribing.
- If your application is private, update the ivim`SubscribeGroup` accordingly.

To modify the IVIM Service configuration, use the SDK`Configuration` object. For more details, refer to the chapter "Configuring the SDK".

Starting & Stopping the IVIM Service

The IVIM service can be started or stopped at any time, provided that the V2X Service is already running. If the V2X Service has not been started, attempting to start the IVIM Service will result in an IllegalStateException.

Example:

```
// Start the IVIM Service
sdk.startIVIMService();

// Stop the IVIM Service
sdk.stopIVIMService();
```

Handling Incoming IVIM Messages

To handle IVIMs emitted by City/Roads Operators which are relevant to your current area, your application needs to subscribe to the event IVIM_LIST_CHANGED.

Example:

```
EventListener myMessagesListener = new EventListener() {
  @Override
  public void onMessageBusEvent(BaseEvent baseEvent) {
     if (baseEvent.getEventType() == EventType.IVIM_LIST_CHANGED) {
          EventIVIMListChanged ilc = (EventIVIMListChanged) baseEvent;
         // Process the IVIM Records
         for (IVIMRecord ivimRecord : ilc.getList()){
                ....
         }
     }
  }
};
sdk.subscribe(myMessagesListener, EventType.IVIM_LIST_CHANGED);
```

8.3.11. SPATEM Service

Configuration

The SPATEM Service handles the reception of Signal Phase and Timing Extended Messages (SPATEMs).

- By default, it uses "public" group for subscribing.
- If your application is private, update the spatemSubscribeGroup accordingly.

To modify the SPATEM Service configuration, use the SDKConfiguration object. For more details, refer to the chapter "Configuring the SDK".

Starting & Stopping the SPATEM Service

The SPATEM service can be started or stopped at any time, provided that the V2X Service is already running. If the V2X Service has not been started, attempting to start the SPATEM Service will result in an IllegalStateException.

Example:

```
// Start the SPATEM Service
 sdk.startSPATEMService();

 // Stop the SPATEM Service
sdk.stopSPATEMService();
```

Handling Incoming SPATEM Messages

To handle SPATEM emitted by City/Roads Operators which are relevant to your current area, your application needs to subscribe to the event SPATEM_LIST_CHANGED.

Example:

```
EventListener myMessagesListener = new EventListener() {
   @Override
   public void onMessageBusEvent(BaseEvent baseEvent) {
       if (baseEvent.getEventType() == EventType.SPATEM_LIST_CHANGED) {
          EventSPATEMListChanged slc = (EventSPATEMListChanged) baseEvent;
         // Process the SPATEM Records
         for (SPATEMRecord spatemRecord : slc.getList()){
          ....
         }
       }
    }
};
sdk.subscribe(myMessagesListener, EventType.SPATEM_LIST_CHANGED);
```

8.3.12. MAPEM Service

Configuration

The MAPEM Service handles the reception of the MAP (topology) Extended Messages (MAPEMs)

- By default, it uses "public" group for subscribing.
- If your application is private, update the mapemSubscribeGroup accordingly.

To modify the MAPEM Service configuration, use the SDKConfiguration object. For more details, refer to the chapter "Configuring the SDK".

Starting & Stopping the MAPEM Service

The MAPEM can be started or stopped at any time, provided that the V2X Service is already running. If the V2X Service has not been started, attempting to start the MAPEM Service will result in an IllegalStateException.

Example:

```
// Start the MAPEM Service
sdk.startMAPEMService();

// Stop the MAPEM Service
sdk.stopMAPEMService();
```

Handling Incoming MAPEM Messages

To handle MAPEM emitted by City/Roads Operators which are relevant to your current area, your application needs to subscribe to the event MAPEM_LIST_CHANGED.

Example:

```
EventListener myMessagesListener = new EventListener() {
   @Override
   public void onMessageBusEvent(BaseEvent baseEvent) {
     if (baseEvent.getEventType() == EventType.MAPEM_LIST_CHANGED) {
       EventMAPEMListChanged mlc = (EventMAPEMListChanged) baseEvent;
       // Process the MAPEM Records
       for (MAPEMRecord mapemRecord : mlc.getList()){
        ....
       }
     }
   }
};
 sdk.subscribe(myMessagesListener, EventType.MAPEM_LIST_CHANGED);
```

8.3.13. VAM Service

Configuration

The VAM Service automatically handles the transmission and reception of VRU Awareness Messages (VAMs).

- By default, it operates in TxAndRx mode, using the "public" group for both publishing and subscribing.
- If your application is private, update the vamSubscribeGroup and vamPublishGroup parameters accordingly.
- To receive only VAM messages from nearby vulnerable road users (VRUs), set the service to RxOnly mode.
- To transmit VAM messages without subscribing to others, configure the service in TxOnly mode.

To modify the VAM Service configuration, use the SDKConfiguration object. For more details, refer to the chapter "Configuring the SDK".

Starting & Stopping the VAM Service

The VAM Service can be started or stopped at any time, provided that the V2X Service is already running. If the V2X Service has not been started, attempting to start the VAM Service will result in an IllegalStateException.

Example:

```
// Start the VAM Service
sdk.startVAMService();

// Stop the VAM Service
sdk.stopVAMService();
```

Handling Incoming VAMs

To handle VAM messages emitted by nearby VRUs, your application needs to subscribe to the event VAM_LIST_CHANGED.

Example:

```
EventListener vamListener = new EventListener() {
  @Override
  public void onMessageBusEvent(BaseEvent baseEvent) {
     if (baseEvent.getEventType() == EventType.VAM_LIST_CHANGED) {
        EventVamListChanged evl = (EventVamListChanged) baseEvent;
        // Process the incoming VAMs here
        List<VAMRecord> vamRecords = evl.getList();
        for (VAMRecord vamRecord : vamRecords) {
           // Implement your logic
           long stationID = vamRecord.getStationID();
           ...
        }
     }
  }
};
sdk.subscribe(vamListener, EventType.VAM_LIST_CHANGED);
```

8.3.14. CPM Service

The CPM Service handles the reception of Collective Perception Message (CPMs)

- By default, it uses "public" group for subscribing.
- If your application is private, update the cpmSubscribeGroup accordingly.

To modify the CPM Service configuration, use the SDKConfiguration object. For more details, refer to the chapter "Configuring the SDK".

Starting & Stopping the CPM Service

The CPM service can be started or stopped at any time, provided that the V2X Service is already running. If the V2X Service has not been started, attempting to start the CPM Service will result in an IllegalStateException.

Example:

```
// Start the CPM Service
sdk.startCPMService();

// Stop the CPM Service
sdk.stopCPMService();
```

Handling Incoming CPM Messages

To handle **CPM** emitted by nearby vehicles or RSU your application needs to subscribe to the Event CPM_LIST_CHANGED

Example:

```
EventListener myMessagesListener = new EventListener() {

  @Override
  public void onMessageBusEvent(BaseEvent baseEvent) {
      if (baseEvent.getEventType() == EventType.CPM_LIST_CHANGED) {
         EventCPMListChanged clc = (EventCPMListChanged) baseEvent;
         // Process the CPM Records
         for (CPMRecord cpmRecord : clc.getList()){
             ....
         }
       }
  }
};
sdk.subscribe(myMessagesListener, EventType.CPM_LIST_CHANGED);
```

8.3.15. POIM Service

Configuration

The POIM Service handles the reception of Point of Interest Message (POIMs).

- By default, it uses "public" group for subscribing.
- If your application is private, update the poimSubscribeGroup accordingly.

To modify the POIM Service configuration, use the SDKConfiguration object. For more details, refer to the chapter "Configuring the SDK".

Starting & Stopping the POIM Service

The POIM service can be started or stopped at any time, provided that the V2X Service is already running. If the V2X Service has not been started, attempting to start the POIM Service will result in an IllegalStateException.

Example:

```
// Start the POIM Service
sdk.startPOIMService();

// Stop the POIM Service
sdk.stopPOIMService();
```

Handling Incoming POIM Messages

To handle **POIM** emitted by City/Roads Operators which are relevant to your current area, your application needs to subscribe to the event POIM_LIST_CHANGED.

Example:

```
EventListener myMessagesListener = new EventListener() {
    @Override
    public void onMessageBusEvent(BaseEvent baseEvent) {
      if (baseEvent.getEventType() == EventType.POIM_LIST_CHANGED) {
         EventPOIMListChanged plc = (EventPOIMListChanged) baseEvent;
         // Process the POIM Records
         for (POIMRecord poimRecord : ilc.getList()) {
            ....
         }
       }
    }
};
sdk.subscribe(myMessagesListener, EventType.POIM_LIST_CHANGED);
```

8.3.16. Custom Messages Service

This service allows to exchange non ETSI messages called Custom Messages with nearby road users.

Configuration

The Custom Message Service automatically handles the transmission and reception of these Custom Messages.

- By default, the service operates in TxAndRx mode, using the "public" group for both publishing and subscribing.
- If you have created a private application, you must update the customMessagesSubscribeGroup and customMessagesPublishGroup parameters accordingly.
- To receive only Custom Messages from nearby vehicles, configure the service in RxOnly mode.
- To transmit Custom messages without subscribing to others, set the service to TxOnly mode.

To modify the Custom Messages Service configuration, use the SDKConfiguration object. For more details, refer to the chapter "Configuring the SDK"

Starting & Stopping the Custom Message Service

The Custom Messages can be started or stopped at any time, provided that the V2X Service is already running. If the V2X Service has not been started, attempting to start the Custom Messages Service will result in an IllegalStateException.

Example:

```
// Start the Custom Messages Service
sdk.startCustomMessagesService();

// Stop the Custom Messages Service
sdk.stopCustomMessagesService();
```

Handling Incoming Custom Messages

To handle Custom Messages emitted by nearby Road Users, your application needs to subscribe to the event INCOMING_CUSTOM_MESSAGE.

Example:

```
EventListener customMessagesListener = new EventListener() {
  @Override
  public void onMessageBusEvent(BaseEvent baseEvent) {
    if (baseEvent.getEventType() == EventType.INCOMING_CUSTOM_MESSAGE) {
      EventIncomingCustomMessage eicl = (EventIncomingCustomMessage) baseEvent;
      // Process the custom Message
      byte[] payload = eicl.getPayload();
    }
  }
};
sdk.subscribe(customMessagesListener, EventType.INCOMING_CUSTOM_MESSAGE);
```

Publish a Custom Message

To publish a Custom Message, call the publishCustomMessage() method and pass the CustomMessage paylaod , the subService name of the CustomMessage and the RefLocation of the CustomMessage as arguments.

```
/**
 * Publish Custom Messages using the CustomMessages Service
 *
 * @param payload    The binary payload of the Custom Message to be sent
 * @param subService The subService name
 * @param refLat     The latitude of the Reference Position linked to this custom message
 * @param refLon     The longitude of the Reference Position linked to this custom message
 * @return true if ALL the parameters are OK and if the message has been sent , else return false.
 * @throws IllegalStateException if V2XService or CustomService is stopped.
 */
```

Example:

```
String myMessage = "hello world";
double msgRefLat=48.866667d;
double msgRefLon=2.333333d;
boolean result = sdk.publishCustomMessage(myMessage.getBytes(),"chatApplication",refLat,refLon);
```

8.3.17. Stop V2X Service

When the application no longer requires the V2XService and needs to free up resources, it can terminate the service using the stopV2XService() method.

Example:

```
// Stop the V2X Service
sdk.stopV2XService();
```

9.Additional Features

9.1. Create your own Custom Dynamic Zone Algorithm

The V2X SDK includes a Dynamic Zone Algorithm that calculates geographic tiles for receiving V2X messages. It computes 9 tiles (your current tile and 8 surrounding tiles) and updates them as you move. This chapter explains how to create your own algorithm and use it for any V2X Service.

First of all, you need to create a class which implements the Interface IDynamicZoneFactory

```
import com.vodafone.v2xsdk4javav2.facade.enums.ServicesSubscriptionMode;
import com.vodafone.v2xsdk4javav2.facade.geoserver.IDynamicZoneFactory;
import com.vodafone.v2xsdk4javav2.facade.geoserver.IDynamicZonesAlgorithm;
import com.vodafone.v2xsdk4javav2.facade.utils.GeoHash;

import java.util.ArrayList;

public class CustomCamZoneAlgorithm implements IDynamicZoneFactory {

   private static final int GEOHASH_LENGTH = 4;

   @Override
   public IDynamicZonesAlgorithm buildZoneAlgorithm() {
      return new IDynamicZonesAlgorithm() {

         private ArrayList<String> geoHashes = new ArrayList<>();

         @Override
         public ArrayList<String> computeGeohashs(ServicesSubscriptionMode serviceSubscriptionMode, double latitude, double
longitude, float speedInKmPerHour, int maxGeohashLength) {
            geoHashes = createOptimisedZones(latitude, longitude);
            return geoHashes;
         }

         @Override
         public Integer getGeohashLength() {
            return geoHashes.size();
         }
      };
   }



   private ArrayList<String> createOptimisedZones(double latitude, double longitude) {
      ArrayList<String> geohashs = new ArrayList<>();
      GeoHash geohash = new GeoHash();
      try {
         String centralGeohash = geohash.encode(latitude, longitude, GEOHASH_LENGTH);
         geohashs.add(centralGeohash);
      } catch (Exception e) {
         e.printStackTrace();
      }
```

```
        return geohashs;
    }
}
```

Once completed, you can use your Custom Dynamic Zone Algorithm with any service. The sample below demonstrates its usage with CAMService.

```
SDKConfiguration configuration = SDKConfiguration.builder()
        .stationID(12345L)
        .stationType(StationType.SPECIAL_VEHICLES)
        .applicationID("APP_ID")
        .applicationToken("APP_TOKEN")
        .camZonesAlgorithmFactory(new CustomCamZoneAlgorithm())
        .build();

// Configure the V2X-SDK
V2XSDK sdk = new V2XSDK(locationProvider,configuration);
```

9.2. Logging

The logger of the V2X SDK is initially disabled by default. To enable it and specify the desired level of logging, use the setSDKLogLevel() method in the V2XSDK class. Available logging modes include OFF, DEBUG, INFO, WARN, and ERROR.

```
// Configure the V2X-SDK
V2XSDK sdk = new V2XSDK(locationProvider,config);
sdk.setSDKLogLevel(LogLevel.DEBUG);
```

10. Annexes

10.1. Events

- FRESH_GPS_NAV
- TIME_SYNC_OK
- ITS_LOCATION_LIST_CHANGED
- CAM_LIST_CHANGED
- DENM_LIST_CHANGED
- IVIM_LIST_CHANGED
- RTT_MEASUREMENT
- V2X_SERVICE_STATE_CHANGED
- V2X_CONNECTIVITY_STATE_CHANGED
- LOCATION_SERVICE_DISABLED
- LOCATION_PERMISSION_NOT_GRANTED
- LP_START
- LP_STOP
- SUBSCRIPTION_AREA_CHANGED
- APPLICATIVE_SERVICE_STATED_CHANGED
- INCOMING_CUSTOM_MESSAGE
- MAPEM_LIST_CHANGED
- SPATEM_LIST_CHANGED
- VAM_LIST_CHANGED
- AREA_LIST_RETRIEVED
- CPM_LIST_CHANGED
- SERVICE_AREA_CHANGED
- INCOMING_TRAFFIC_MEASUREMENT
- ACTIVE_DENM_CHANGED
- ICC_STATUS_CHANGED

See the Java documentation for more details.

10.2. Records

- CAMRecord
- CPMRecord
- DENMRecord
- IVIMRecord
- MAPEMRecord
- SPATEMRecord
- VAMRecord
- ITSLocationRecord
- POIMRecord

See the Java documentation for more details.

10.3. FakeGNSSReceiver Class

```
/**
* A fake GNSS receiver for testing purposes.
*/ public class FakeGNSSReceiver extends LocationProvider implements Runnable {
public final String TAG = DynamicTAG.getDynamicTag("FakeGNSSReceiver");
private final Object lock = new Object();
private boolean isRunning = false;
private final double fakeLatitude;
private final double fakeLongitude;

/**
 * Constructs a FakeGNSSReceiver using the specified fake latitude and longitude.
 *
 * @param fakeLatitude  the fake latitude
 * @param fakeLongitude the fake longitude
 */
public FakeGNSSReceiver(double fakeLatitude, double fakeLongitude) {
   super();
   this.fakeLatitude = fakeLatitude;
   this.fakeLongitude = fakeLongitude;
}

/**
 * Turns on the fake GNSS receiver.
 *
 * @return true if the receiver was successfully turned on, false otherwise
 */
@Override
protected boolean turnOn() {
   Logger.debug(TAG, "turnOn()");
   Thread thread = new Thread(this);
   thread.start();
   return true;
}

/**
 * Turns off the fake GNSS receiver.
 */
@Override
protected void turnOff() {
   Logger.debug(TAG, "turnOff()");
   synchronized (lock) {
      isRunning = false;
      lock.notifyAll();
   }
}

/**
 * Runs the fake GNSS receiver, generating location updates at regular intervals.
 */
@Override
public void run() {
   isRunning = true;
   Logger.debug(TAG, TAG + " Thread started");
   while (isRunning) {
      GnssLocation fixedLocation = new GnssLocation(fakeLatitude, fakeLongitude, 450d, 90f, 10f, null, UTCTimeProvider.getUTCTimeInMs());
      notifyFreshLocation(fixedLocation);
      try {
         synchronized (lock) {
            lock.wait(1000L);
         }
      } catch (InterruptedException e) {
         Logger.error(TAG, "E53", e);
         Thread.currentThread().interrupt();
```

```
        }
    }
    Logger.debug(TAG, TAG + " Thread stopped");
}

}
```