



Evaluation and validation of connected
mobility in real open systems beyond
5GS

Project deliverable D3.2

ENVELOPE open and dynamically reconfigurable B5GS description

HORIZON JU Innovation Actions | 101139048 |
ENVELOPE - HORIZON-JU-SNS-2023



Co-funded by
the European Union

6GSNS

Deliverable administrative information

Dissemination level	PU - Public
Type of deliverable	R — Document, report
Work package	WP3
Title of the deliverable	D3.2 – ENVELOPE open and dynamically reconfigurable B5GS description
Status – version, date	Final – V1.0, 17/10/2025
Deliverable leader	NCSR
Contractual date of delivery	30/09/2025
Submission date	17/10/2025
Keywords	API implementation, Native APIs, CAMARA APIs, Beyond 5G System

List of contributors

Name	Organisation
Dimitris Uzunidis	NCSR
Harilaos Koumaras	NCSR
Vasilis Pitsilis	NCSR
Edoardo Bonetto	LINKS
Ramon de Souza Schwartz	TNO
Nikolaos Petropouleas	LNVO
Apostolis Salkintzis	LNVO
Christos Milarokostas	FOGUS
Alex Kakyris	FOGUS
Fabrizio Gatti	TIM

Davide Montagno Bozzone	HPE
Nicola di Pietro	HPE
Gabriele Scivoletto	NXW
Giacomo Bernini	NXW

Quality control

	Name	Organisation	Date
Peer review 1	Daniele Brevi	LINKS	6/10/2025
Peer review 2	Ramon de Souza Schwartz	TNO	6/10/2025

Version History

Version	Date	Author	Summary of changes
0.1	15/06/2025	Dimitris Uzunidis	Initial ToC
0.2	15/09/2025	Dimitris Uzunidis	Final ToC
0.3	29/09/2025	All	First Round of Contributions
0.4	6/10/2025	Dimitris Uzunidis, Harilaos Koumaras	First Round of Contributions Reviewed by Deliverable editor
0.5	6/10/2025	TNO, LINKS	Internal Review by two reviewers
0.6	10/10/2025	All	Revisions addressed by all contributors
0.7	13/10/2025	TNO, LINKS	Final Review by two reviewers
0.8	15/10/2025	Dimitris Uzunidis, Harilaos Koumaras	Final Review by Deliverable editor
0.9	16/10/2025	Konstantinos Katsaros, Anargyros Roumeliotis	PM Review and Quality Check
1.0	17/10/2025	Konstantinos Katsaros	Deliverable Submitted

Legal Disclaimer

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or Smart Networks and Services Joint Undertaking (SNS JU). Neither the European Union nor SNS JU can be held responsible for them.

Copyright © ENVELOPE Consortium, 2024.

Table of contents

DELIVERABLE ADMINISTRATIVE INFORMATION	I
TABLE OF CONTENTS	IV
LIST OF FIGURES	VIII
LIST OF TABLES	X
PROJECT EXECUTIVE SUMMARY	XI
DELIVERABLE EXECUTIVE SUMMARY	XII
LIST OF ABBREVIATIONS AND ACRONYMS	XIII
1 INTRODUCTION	16
1.1 Purpose of the deliverable	16
1.2 Relation with Deliverable 3.3	16
1.3 Intended audience	18
1.4 Structure of the deliverable	18
2 ENVELOPE PLATFORM OVERVIEW	19
2.1 ENVELOPE Framework Overview	19
2.2 Southbound and Northbound APIs	20
2.3 B5G network functionalities abstraction enablers - Quality on Demand enabler, Device Location enablers, Performance metric enabler	21
2.3.1 Quality on Demand (QoD) Enabler	21
2.3.2 Device Location Enabler	22
2.3.3 Performance Metric Enabler	22
3 NATIVE APIS DESIGN AND IMPLEMENTATION	24
3.1 AsSessionWithQoS API	24
3.1.1 Short description	24
3.1.2 Relation with standards	24
3.1.3 Functional overview	24

3.1.4	<i>API endpoints</i>	26
3.1.5	<i>Request and response examples</i>	27
3.1.6	<i>Error codes & error handling</i>	28
3.2	<i>MonitoringEvent API</i>	28
3.2.1	<i>Short description</i>	28
3.2.2	<i>Relation with standards</i>	28
3.2.3	<i>Functional overview</i>	28
3.2.4	<i>API endpoints</i>	29
3.2.5	<i>Request and response examples</i>	30
3.2.6	<i>Error codes & error handling</i>	31
3.3	<i>MEC Location API</i>	31
3.3.1	<i>Short description</i>	31
3.3.2	<i>Relation with standards</i>	32
3.3.3	<i>Functional overview</i>	32
3.3.4	<i>API endpoints</i>	34
3.3.5	<i>Request and Response examples</i>	36
3.3.6	<i>Error codes & error handling</i>	38
3.4	<i>AnalyticsExposure API</i>	38
3.4.1	<i>Short description</i>	38
3.4.2	<i>Relation with standards</i>	39
3.4.3	<i>Functional overview</i>	39
3.4.4	<i>API endpoints</i>	39
3.4.5	<i>Request and response examples</i>	40
3.4.6	<i>Error codes & error handling</i>	41
4	ENVELOPE APIS DESIGN AND IMPLEMENTATION	42
4.1	<i>Quality on Demand (QoD) API</i>	42
4.1.1	<i>Short description</i>	42
4.1.2	<i>Relation with standards</i>	42
4.1.3	<i>Functional overview</i>	42
4.1.4	<i>API endpoints</i>	47
4.1.5	<i>Request and response examples</i>	49
4.1.6	<i>Error codes & error handling</i>	51

4.2	<i>Location Retrieval API</i>	51
4.2.1	<i>Short description</i>	51
4.2.2	<i>Relation with standards</i>	51
4.2.3	<i>Functional overview</i>	51
4.2.4	<i>API endpoints</i>	52
4.2.5	<i>Request and response examples</i>	53
4.2.6	<i>Error codes & error handling</i>	54
4.3	<i>Device Location (Geofencing Subscriptions) API</i>	54
4.3.1	<i>Short description</i>	54
4.3.2	<i>Relation with standards</i>	54
4.3.3	<i>Functional overview</i>	54
4.3.4	<i>API endpoints</i>	55
4.3.5	<i>Request and response examples</i>	56
4.3.6	<i>Error codes & error handling</i>	58
4.4	<i>Devices In Area API</i>	58
4.4.1	<i>Short description</i>	58
4.4.2	<i>Relation with standards</i>	58
4.4.3	<i>Functional overview</i>	59
4.4.4	<i>API endpoints</i>	60
4.4.5	<i>Request and response examples</i>	61
4.4.6	<i>Error codes & error handling</i>	63
4.5	<i>Performance Metrics API</i>	63
4.5.1	<i>Short description</i>	63
4.5.2	<i>Relation with standards</i>	64
4.5.3	<i>Functional overview</i>	64
4.5.4	<i>API endpoints</i>	65
4.5.5	<i>Request and response examples</i>	65
4.5.6	<i>Error codes & error handling</i>	66
5	INTEGRATION OF APIS INTO THE TRIAL SITES	67
5.1	<i>Italian Site - Enabler Mapping and Specific Implementations</i>	67
5.2	<i>Dutch Site - Enabler Mapping and Specific Implementations</i>	68
5.3	<i>Greek Site - Enabler Mapping and Specific Implementations</i>	69

6	NEXT STEPS	71
7	CONCLUSIONS	72

List of figures

Figure 1 – The revised ENVELOPE architecture that updates the version presented in D2.1.	20
Figure 2 – Sequence Diagram AsSessionWithQoS APIs.	25
Figure 3 – Subscription endpoints based on scsAsId.	26
Figure 4 – Subscription endpoints based on scsAsId and subscriptionId.	26
Figure 5 – Sequence diagram for MonitoringEvent API for location reporting.	29
Figure 6 – MonitoringEvent <i>subscriptions</i> endpoints.	29
Figure 7 – MonitoringEvent subscription-specific endpoints.	30
Figure 8 – Sequence diagram for MEC013 – User Area Subscribe API.	33
Figure 9 – Sequence diagram for MEC013 – User Discovery Subscribe API.	34
Figure 10 – User Area Subscribe <i>subscriptions</i> endpoints.	34
Figure 11 – User Area Subscribe subscription-specific endpoints.	35
Figure 12 – User Discovery <i>subscriptions</i> endpoints.	35
Figure 13 – User Discovery subscription-specific endpoints.	36
Figure 14 – Sequence diagram for AnalyticsExposure API.	39
Figure 15 – AnalyticsExposure endpoints.	39
Figure 16 – Sequence Diagram for QoD Retrieve available QoS Profiles request API.	43
Figure 17 – Sequence Diagram for QoD session request API.	43
Figure 18 – Sequence Diagram for QoD get individual session info API.	44
Figure 19 – Sequence Diagram for QoD Delete individual session info API.	45
Figure 20 – Sequence Diagram for QoD Retrieve all QoS sessions API.	46
Figure 21 – Sequence Diagram for QoD Extension of a QoS session API.	46
Figure 22 – Swagger UI for QoD APIs.	47
Figure 23 – Message flow for Device location (Location Retrieval) API.	52
Figure 24 – Location Retrieval API endpoint.	52
Figure 25 – MonitoringEvent API endpoint as southbound communication.	53
Figure 26 – Sequence diagram the Device Location (Geofencing) API.	55
Figure 27 – Geofencing subscription endpoints.	55
Figure 28 – Geofencing subscription-specific endpoints.	56
Figure 29 – Sequence diagram of the Devices in Area Query API.	59
Figure 30 – Sequence diagram the Devices in Area Subscription API.	60
Figure 31 – Devices in Area query endpoint.	60

Figure 32 – Devices in Area subscription endpoints.	61
Figure 33 – Devices in Area subscription-specific endpoints.	61
Figure 34 – Message flow for the Performance Metrics API.	64
Figure 35 – Performance Metrics subscription endpoints.	65
Figure 36 – Performance Metrics subscription-specific endpoints.	65
Figure 37 – Overall system architecture in the Italian site.	67
Figure 38 – Overall system architecture in the Dutch site.	68
Figure 39 – Overall system architecture in the Greek site.	70

List of tables

Table 1 – ENVELOPE API Distribution Across Deliverables 3.2 and 3.3.....	17
Table 2 – Native and ENVELOPE APIs included in this Deliverable and are related with fundamental network exposure capabilities.....	21
Table 3 – Native and ENVELOPE APIs defined for each enabler used in the Italian site.	68
Table 4 – Native and ENVELOPE APIs defined for each enabler used in the Dutch site.....	69
Table 5 – Native and ENVELOPE APIs defined for each enabler used in the Greek site.	70

Project executive summary

ENVELOPE aims to advance and open the reference 5G (5th Generation) advanced architecture and transform it into a vertical-oriented one. It proposes a novel open and easy-to-use 5G-advanced architecture to enable a tighter integration of the network and the service information domains by

- exposing network capabilities to verticals,
- providing vertical information to the network; and
- enabling verticals to dynamically request and modify key network aspects,
- all performed in an open, transparent, and easy-to-use, semi-automated way.

ENVELOPE will build Application Programming Interfaces (APIs) that act as an intermediate abstraction layer that translates the complicated 5GS interfaces and services into easy to consume services accessible by the vertical domain. The experimentation framework and the main innovations developed in the project are: Multi-access Edge Computing (MEC) with service continuity support, zero-touch management, multi-connectivity and predictive Quality of Service (QoS).

It will deliver 3 large scale Beyond 5G (B5G) trial sites in Italy, Netherlands and Greece supporting novel vertical services, with advanced exposure capabilities and new functionalities tailored to the services' needs. Although focused on the Connected and Automation Mobility (CAM) vertical, the developments resulting from the use cases (UC) will be reusable by any vertical. The ENVELOPE architecture will serve as an envelope that can cover, accommodate, and support any type of vertical services. The applicability of ENVELOPE will be demonstrated and validated via the project CAM UCs and via several 3rd parties that will have the opportunity to conduct funded research and test their innovative solutions over ENVELOPE thanks to the project open calls.

Social Media link:



[@envelope-project](https://www.linkedin.com/company/envelope-project)

For further information please visit www.envelope-project.eu

Deliverable executive summary

Work Package 3 (WP3) is responsible for developing the activities related to the implementation of the ENVELOPE platform and its associated innovations. The ENVELOPE platform is a B5G System Experimentation as a Service (EaaS) platform that provides simplified and extended interactions with the 5G network. Task 3.2 “Developments towards an open and dynamically reconfigurable B5GS (cellular system Beyond 5GS)” focuses mainly on i) the “Native” APIs that will allow the technology enablers, developed and implemented within ENVELOPE, to interact with the B5G network and ii) the “simplified” APIs, named as “ENVELOPE APIs” hereafter, exposed by the ENVELOPE enablers and consumed by the CAM applications to abstract network complexities and integrate network and edge capabilities.

This deliverable documents the outcomes of Task 3.2, which are the development and implementation of

- Native and ENVELOPE APIs about the QoS and Quality on Demand (QoD), respectively;
- Native APIs about user’s location (MonitoringEvent API and MEC Location API), and corresponding ENVELOPE APIs about user’s location (Location Retrieval API, Device Location API - Geofencing Subscriptions and Devices in Area API); as well as
- Native and ENVELOPE APIs about performance, particular the *AnalyticsExposure* and Performance Metrics (Network Data Analytics Function - NWDAF), respectively.

The next Deliverable, D3.3, will build on this one to document the APIs focused on CAM service development, including the edge cloud continuum, the predictive QoS, the MEC handover and the Access Traffic Steering, Switching and Splitting (ATSSS) enablers. This segregation is performed to create a logical division, where D3.2 focuses on network exposure APIs (QoS/QoD, location services and performance metrics) that provide essential 5G network programmability building blocks, while D3.3 addresses more service-oriented APIs for advanced CAM functionalities. All those APIs will be exploited by the three trials sites and will feed Tasks 4.1 - 4.4 which will document their integration and pre-evaluation testing in the corresponding deliverables.

List of abbreviations and acronyms

Acronym	Meaning
3GPP	3rd Generation Partnership Project
5G	5th Generation
5QI	5G QoS Identifier
AF	Application Function
AMF	Access and Mobility Management Function
API	Application Programming Interface
AS	Application Server
ATSSS	Access Traffic Steering, Switching, and Splitting
B5G	Beyond 5G
B5GS	Beyond 5G System
BSF	Binding Support Function
CAM	Cooperative, Connected, and Automated Mobility
CAPIF	Common API Framework
DT	Digital Twin
EC	European Commission
ETSI	European Telecommunications Standards Institute
EaaS	Experimentation as a Service
GMLC	Gateway Mobile Location Centre
GNSS	Global Navigation Satellite System
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
MEC	Multi-access Edge Computing

MQTT	Message Queuing Telemetry Transport
MSISDN	Mobile Station International Subscriber Directory Number
NEF	Network Exposure Function
NGAP	NG Application Protocol
NWDAF	Network Data Analytics Function
OBU	On-Board Unit
PCF	Policy Control Function
PDU	Protocol Data Unit
PF	Policy Function
PLMN	Public Land Mobile Network
PU	Public
QoD	Quality on Demand
QoS	Quality of Service
RAN	Radio Access Network
SM	Session Management
SMF	Session Management Function
SNS JU	Smart Networks and Services Joint Undertaking
UC	Use Case
UE	User Equipment
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UUID	Universally Unique Identifier
V2I	Vehicle-to-Infrastructure
V2V	Vehicle-to-Vehicle

V2X	Vehicle-to-Everything
WP	Work Package
pQoS	Predictive Quality of Service
UPF	User Plane Function

1 Introduction

1.1 Purpose of the deliverable

This deliverable documents the outcomes of activities conducted within WP3, specifically focusing on the development and implementation of the B5G system capabilities that enable the ENVELOPE platform's network exposure functionality. The primary objective is to present the implementation of the APIs for the open and dynamically reconfigurable B5G system across three trial sites in Italy, Netherlands and Greece.

This document includes the foundational API infrastructure and enabler implementations that bridge the complexity gap between advanced 5G network functions and vertical industry applications, particularly in the CAM domain. The technical developments reported here establish the essential network exposure capabilities that enable simplified access to sophisticated B5G functionalities through standardized, developer-friendly interfaces.

To achieve this objective, the activities of this deliverable focus on three main aspects: first, the implementation of fundamental network exposure APIs including QoD, Device Location services and Performance Metrics enablers, second, the detailed specification of the API abstraction framework that transforms complex southbound Native APIs into simplified northbound ENVELOPE APIs and third, the establishment of the architectural foundation that supports interoperable deployment across diverse trial site infrastructures.

The development approach documented here builds upon the platform architecture introduced in ENVELOPE deliverable D2.2 and the EaaS module implementation detailed in D3.1, providing the essential building blocks for advanced service-oriented capabilities planned for subsequent deliverables. This foundational layer serves as a prerequisite for implementing sophisticated functionalities, such as predictive analytics, cross-domain service migration and multi-connectivity features that will be documented in future project outputs.

1.2 Relation with Deliverable 3.3

As ENVELOPE consortium we performed a logical separation of the APIs between i) those focused on fundamental network exposure capabilities, which are the ones related with QoS, Device Location and Performance Metrics, and ii) those related to advanced service-oriented capabilities including Predictive Quality of Service (pQoS), ATSSS multi-connectivity, MEC handover and edge computing continuum management. The former are included and analyzed in this deliverable while the latter will be analyzed in the next WP3 deliverable (D3.3). The APIs presented here provide the essential infrastructure for any 5G/B5G network interaction as they establish basic connectivity, monitoring and location awareness and serve as prerequisites for more sophisticated functionalities.

The main rationale behind this segregation within ENVELOPE is to follow the natural application development patterns where basic network access must be established before implementing exploitation of predictive analytics or cross-domain service migration. The division ensures focused deliverable scope while enabling seamless integration of increasingly sophisticated B5G innovations essential for connected mobility deployments. Each deliverable maintains clear

boundaries while building towards ENVELOPE platform capabilities that support autonomous vehicle operations requiring both fundamental network access and advanced orchestration features. To facilitate understanding of this API distribution and provide clear reference for implementation activities, we tabulate in Table 1 the mapping of all ENVELOPE APIs, categorizing them by their nature (Native or CAMARA-compliant/CAMARA-style) and indicating their respective deliverable allocation.

Table 1 – ENVELOPE API Distribution Across Deliverables 3.2 and 3.3.

API name	Category	Deliverable
Quality of Service (QoS) / Quality of Demand (QoD) APIs		
AsSessionWithQoS	Native	3.2
Quality on Demand (QoD)	CAMARA	3.2
Device Location APIs		
MonitoringEvent	Native	3.2
MEC Location	Native	3.2
Location Retrieval	CAMARA	3.2
Device Location (Geofencing Subscriptions)	CAMARA	3.2
Devices In Area	CAMARA	3.2
Performance Metrics APIs		
AnalyticsExposure	Native	3.2
Performance Metrics (NWDAF)	CAMARA	3.2
Predicted Quality of Service (pQoS) APIs		
<ul style="list-style-type: none"> AnalyticsSubscription AnalyticsInfo Request 	Native	3.3
<ul style="list-style-type: none"> PQoS-Inference Analytics Variants Analytics Exposure Modes Supported QoS Types 	CAMARA	3.3
Access Traffic Steering, Switching, and Splitting (ATSSS) APIs		
<ul style="list-style-type: none"> ATSSS-Cu ATSSS-Cp 	Native	3.3
<ul style="list-style-type: none"> ATSSS-T 	ETSI MEC-015 inspired	3.3
Edge computing APIs		
<ul style="list-style-type: none"> ETSI MEC GS 013 APIs aerOS APIs 	Native	3.3
<ul style="list-style-type: none"> Kubernetes / Helm APIs 	Native	
<ul style="list-style-type: none"> Edge Cloud (Edge Application Management) APIs 	CAMARA	3.3

1.3 Intended audience

The dissemination level of D3.2 is public (PU), making this deliverable available to consortium members, European Commission (EC) Services and external stakeholders. This document primarily serves as a technical reference for all ENVELOPE beneficiaries, particularly partners involved in WP3 technical innovation activities and WP4 trial site integration efforts.

The document is designed for multiple audiences including project partners implementing ENVELOPE enablers and APIs, trial site operators deploying the B5G system infrastructure, vertical application developers who will consume the exposed APIs, external experimenters participating in ENVELOPE Open Calls, standardization bodies interested in B5G network exposure approaches and the broader research community working on B5G network programmability and vertical industry integration.

Additionally, this deliverable serves as a key reference for understanding how complex 3rd Generation Partnership Project (3GPP) network functions can be abstracted into simplified, standards-compliant interfaces that enable innovation in vertical domains without requiring deep telecommunications expertise.

1.4 Structure of the deliverable

The structure of this deliverable is organized as follows: Section 2 introduces the core ENVELOPE API framework, elucidating how essential network exposure features are enabled through dedicated enablers for QoD, Device Location and Performance Metrics. This section also establishes the underlying architectural concepts that facilitate the transformation from Native APIs to the simplified, developer-friendly ENVELOPE APIs tailored for vertical integration.

Sections 3 and 4 present the technical specifications of each API associated with its corresponding enablers. It covers the design, relation with standards, functional overview, API endpoints, request and response examples as well as error handling. Through this, it is demonstrated how ENVELOPE APIs are capable of addressing unique requirements of diverse vertical applications within the three pilot environments.

Section 5 focuses on the practical deployment, offering site-specific details for each trial installation. It presents how the common ENVELOPE API framework is adapted and instantiated within varying infrastructure conditions, while ensuring uniform and consistent interfaces towards application developers and service providers.

Section 6 outlines the immediate next steps and planned developments, including the linkage between the foundational capabilities documented in this deliverable and the more advanced service-oriented functionalities for future project outputs. Finally, section 7 summarizes the key achievements, technical lessons learned and identified limitations from implementing the network exposure APIs. It synthesizes the outcomes across the three enablers and trial sites, while outlining the logical progression toward advanced service-oriented capabilities documented in D3.3 and future deliverables.

2 ENVELOPE Platform Overview

2.1 ENVELOPE Framework Overview

The ENVELOPE project advances the integration of B5G networks with vertical industries, particularly focusing on information exposed to the CAM applications. This is made possible through the ENVELOPE enablers which are software components that first consume southbound (Native) APIs (Far-Edge APIs, Edge APIs and B5G Network APIs) through Common API Framework (CAPIF) to interact with the underlying 5G network infrastructure and edge deployments capabilities. Then, these enablers expose northbound, simplified (ENVELOPE) APIs which abstract network complexities and combine network functionalities with edge capabilities to provide easy-to-consume interfaces, easily accessible by vertical services. ENVELOPE's architecture has been established and elaborated in the Deliverable 2.2 while a revised version has been presented in Deliverable 3.1 and also shown in Figure 1.

Regarding the adopted terminology, we clarify that the term "Native" in ENVELOPE is used to denote the technology-specific APIs and interfaces exposed by underlying B5G network functions and edge platforms. Within this deliverable, "Native APIs" refers exclusively to the underlying, vendor- and standards-specific network interfaces (e.g., 3GPP Network Exposure Function (NEF)/NWDAF, European Telecommunications Standards Institute (ETSI) MEC) that serve as southbound endpoints. This terminology is distinct from common usage of "native" in cloud-native or AI-native contexts. ENVELOPE enablers consume these Native APIs and abstract their complexity into simplified northbound ENVELOPE APIs for vertical application consumption.

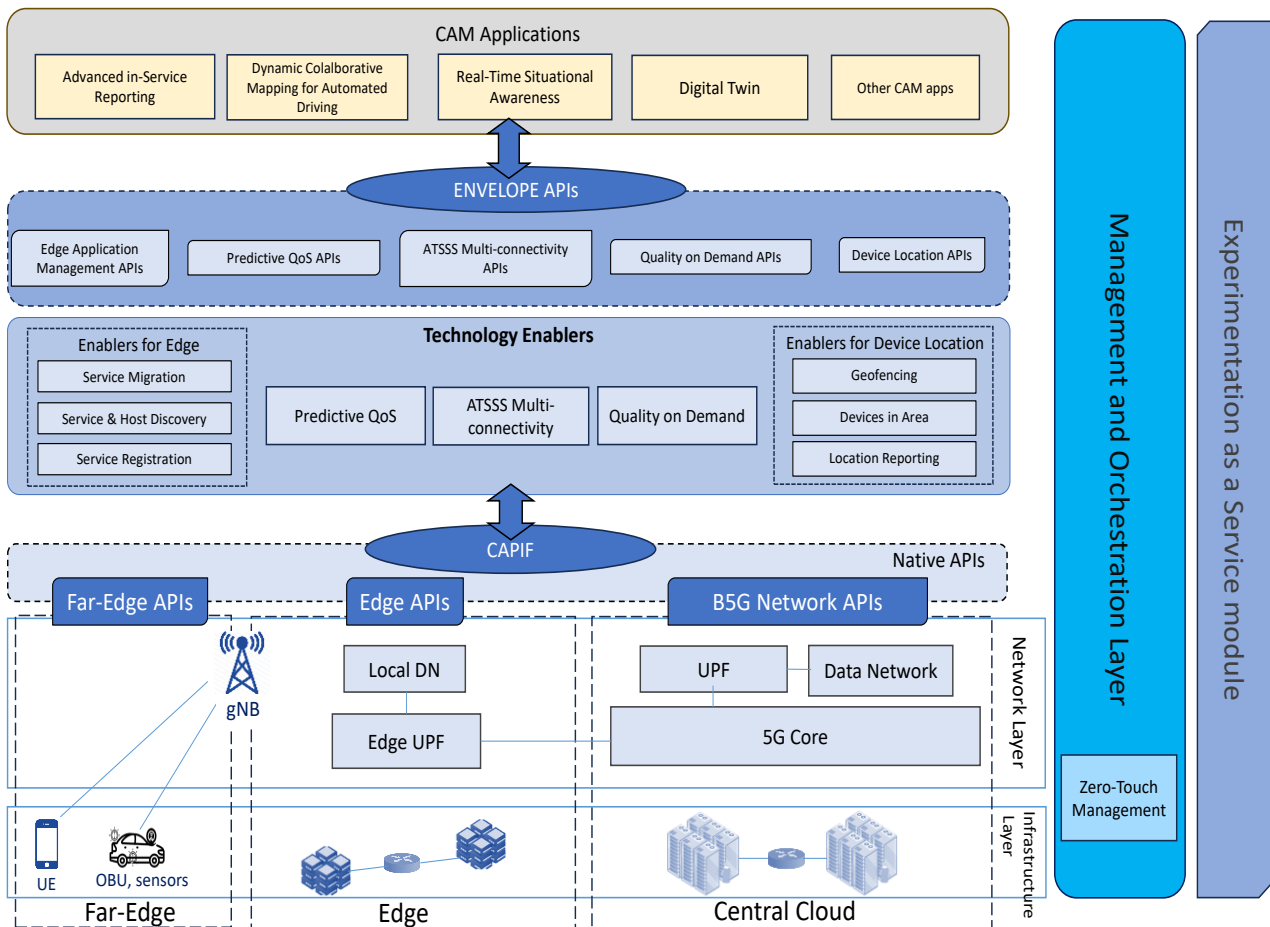


Figure 1 – The revised ENVELOPE architecture that updates the version presented in D2.1.

2.2 Southbound and Northbound APIs

The main details of the APIs that are focused on fundamental network exposure capabilities and are included in this deliverable are tabulated in Table 2. According to this table, the QoS APIs represent the core of dynamic network management, combining the Native 3GPP *AsSessionWithQoS* API for standards-compliant network interaction with the CAMARA QoD API for simplified access. These APIs are exploited by the QoD enabler so that vertical applications can request, in a simplified way, guaranteed throughput or consistent latency without needing deep knowledge of 5G technical specifications. For automotive use cases, this becomes particularly important where millisecond delays can affect safety in collision prevention or autonomous driving scenarios.

Next, Device Location APIs demonstrate the ENVELOPE's approach to location-based services through multiple complementary interfaces. The Native 3GPP *MonitoringEvent* API handles cell-ID based location reporting, while the ETSI MEC Location APIs provide more precise positioning capabilities. The northbound side includes CAMARA Location Retrieval for basic position queries, CAMARA Device Location Geofencing Subscriptions for area-based alerts and the innovative "Devices in Area" API specifically created for identifying vehicles within geographical regions during emergencies or collaborative mapping scenarios.

Finally, Performance Metrics APIs through the performance enabler transform complex NWDAF information through 3GPP *AnalyticsExposure* APIs into more simplified Performance Metrics interfaces. This enables applications to access real-time network analytics including uplink/downlink measurements and communication patterns without directly interacting with 5G core functions. For Digital Twin (DT) applications in particular, this provides critical monitoring capabilities for data-driven resource allocation decisions.

Table 2 – Native and ENVELOPE APIs included in this Deliverable and are related with fundamental network exposure capabilities.

API name	Category	Standards Compliance	Link	Section
Quality of Service (QoS) / Quality of Demand (QoD) APIs				
AsSessionWithQoS	Native	3GPP NEF compliant (3gpp-as-session-with-qos)	OpenAPI	3.1
Quality on Demand (QoD)	CAMARA	CAMARA compliant v1.0.0 (https://camaraproject.org/quality-on-demand/)	OpenAPI	4.1
Device Location APIs				
MonitoringEvent	Native	3GPP NEF compliant (3gpp-monitoring-event) – cell ID	OpenAPI	3.2
MEC Location	Native	ETSI GS MEC 013 (MEC Location API)	OpenAPI	3.3
Location Retrieval	CAMARA	CAMARA compliant (https://camaraproject.org/location-retrieval/)	OpenAPI	4.2
Device Location (Geofencing Subscriptions)	CAMARA	CAMARA compliant v0.3.0 (https://camaraproject.org/geofencing-subscriptions/)	OpenAPI	4.3
Devices In Area	CAMARA	New API will be implemented - following the CAMARA Device Location Geofencing Subscriptions approach	ENVELOPE specs doc	4.4
Performance Metrics APIs				
AnalyticsExposure	Native	3GPP NEF compliant	OpenAPI	3.4
Performance Metrics (NWDAF)	CAMARA	New API will be implemented - closest API is the "Session Insights" API	ENVELOPE specs doc	4.5

2.3 B5G network functionalities abstraction enablers - Quality on Demand enabler, Device Location enablers, Performance metric enabler

2.3.1 Quality on Demand (QoD) Enabler

The QoD enabler consisting of one or more software components enables developers and API consumers to request specific service levels, such as guaranteed throughput or consistent low latency, for their applications' data streams. The QoD enabler abstracts the complexities of underlying 5G/B5G network technologies eliminating the need for vertical applications to master detailed network configurations while ensuring predictable network performance across all three ENVELOPE trial sites. Within the ENVELOPE framework, the QoD enabler allows application functions to achieve stable latency in vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communications. This capability is important for use cases such as autonomous driving and collision prevention, where significant delays or jitter, e.g. few milliseconds, can significantly impact safety outcomes. The enabler operates by consuming southbound 3GPP *AsSessionWithQoS*

Native APIs and exposing northbound CAMARA-compliant QoD API¹. This abstraction layer provides simplified session management operations including creation, retrieval and deletion of QoS sessions, enabling applications to dynamically adjust network quality parameters based on real-time requirements.

2.3.2 Device Location Enabler

The Device Location enabler incorporates multiple location-aware functionalities that allow vertical applications to access and subscribe to device positioning information through standardized interfaces. This enabler integrates 3GPP *MonitoringEvent* APIs and ETSI MEC Location APIs to provide location services, such as geofencing subscriptions, location retrieval and area-based device discovery. The enabler's geofencing functionality allows vertical applications to subscribe to event-based notifications when specified devices enter or leave defined geographical areas. This capability can support various CAM services requiring tracking of asset or user movements, enabling service providers to trigger other APIs, edge-cloud services, etc., based on location updates. The Location Retrieval functionality for the Greek site, enables applications to query the last known location of mobile devices as detected by the mobile network operator, supporting use cases including service personalization and emergency services. The API used is the CAMARA Location Retrieval API². For the Italian trial site, the Device Location enabler implements a specialized "Devices in Area" API that provides information about vehicles within specific geographical regions, supporting both advanced in-service reporting and dynamic collaborative mapping use cases. The Dutch site utilizes Device Location Geofencing Subscriptions³ that comply with CAMARA specifications, enabling real-time alerts for logistics and asset tracking applications. The enabler operates through transformation functions that convert 3GPP location notifications (e.g., cell ID) into simplified Device Location notifications (area-entered, area-left), providing standardized interfaces regardless of the underlying network implementation.

2.3.3 Performance Metric Enabler

The Performance Metric enabler provides real-time network analytics and performance monitoring capabilities by exposing simplified interfaces to access NWDAF information. This enabler consumes southbound 3GPP *AnalyticsExposure* APIs and transforms complex network analytics into easily consumable performance metrics for vertical applications, enabling data-driven decision making and proactive network management. The enabler allows Application Functions (AFs) to access network analytics generated by NWDAF through the NEF, allowing third-party applications to subscribe to, receive and manage analytics information securely without direct interaction with complex 5G core functions. Key performance metrics include: uplink and downlink volume measurements, throughput monitoring and communication analytics that support pQoS decisions and network optimization. This enabler is particularly valuable for DT applications in the Dutch trial site, where Vehicle-to-Everything (V2X) services consume Performance Metrics APIs to monitor achieved uplink and downlink throughput performance, ensuring optimal data collection and mixed reality testing operations. The real-time monitoring capabilities enable proactive network

¹ <https://camaraproject.org/quality-on-demand/>

² <https://camaraproject.org/location-retrieval/>

³ <https://camaraproject.org/geofencing-subscriptions/>

management and support data-driven decision making for dynamic resource allocation across ENVELOPE use cases. The Performance Metric enabler implements periodic fetching of analytics information related to User Equipment (UE) communication, focusing on downlink and uplink volume metrics that are essential for maintaining service quality and enabling predictive network behaviors.

3 Native APIs Design and Implementation

3.1 AsSessionWithQoS API

3.1.1 Short description

The *AsSessionWithQoS* API enables an AF to request from the network to provide specific QoS to an application session for one UE or for a list of UEs, in accordance with 3GPP TS 29.122 and 3GPP TS 23.502. The AF typically specifies the required QoS level by indicating a 5QI (5G QoS Identifier) value or specific QoS parameters such as guaranteed and maximum bit rates. This RESTful API defines the data models, resources and procedures for creating and managing Application Server (AS) sessions with required QoS (3GPP TS 29.122, subclause 5.14.1). The session setup and policy enforcement are implemented via the Policy Control Function (PCF)'s *Npcf_PolicyAuthorization* standard call, with the NEF mediating the request to the Policy Function (PF). This API is important for the purposes of ENVELOPE project as it particularly allows CAM applications to request from the B5G network to apply specific QoS for their application traffic.

3.1.2 Relation with standards

The *AsSessionWithQoS* API developed in the project complies with 3GPP NEF compliant (3gpp-as-session-with-qos) specification. It particularly exploits 3GPP TS 29.122 V17.2.0 T8 reference point for Northbound APIs⁴, 3GPP TS 29.571 Common Data Types for Service Based Interfaces, version 17.2.0⁵ and 3GPP TS 29.514 V17.1.0; 5G System; Policy Authorization Service; Stage 3⁶.

3.1.3 Functional overview

Figure 2 illustrates the sequence diagram of the implemented *AsSessionWithQoS* API.

⁴ http://www.3gpp.org/ftp/Specs/archive/29_series/29.122/

⁵ http://www.3gpp.org/ftp/Specs/archive/29_series/29.571/

⁶ http://www.3gpp.org/ftp/Specs/archive/29_series/29.514/

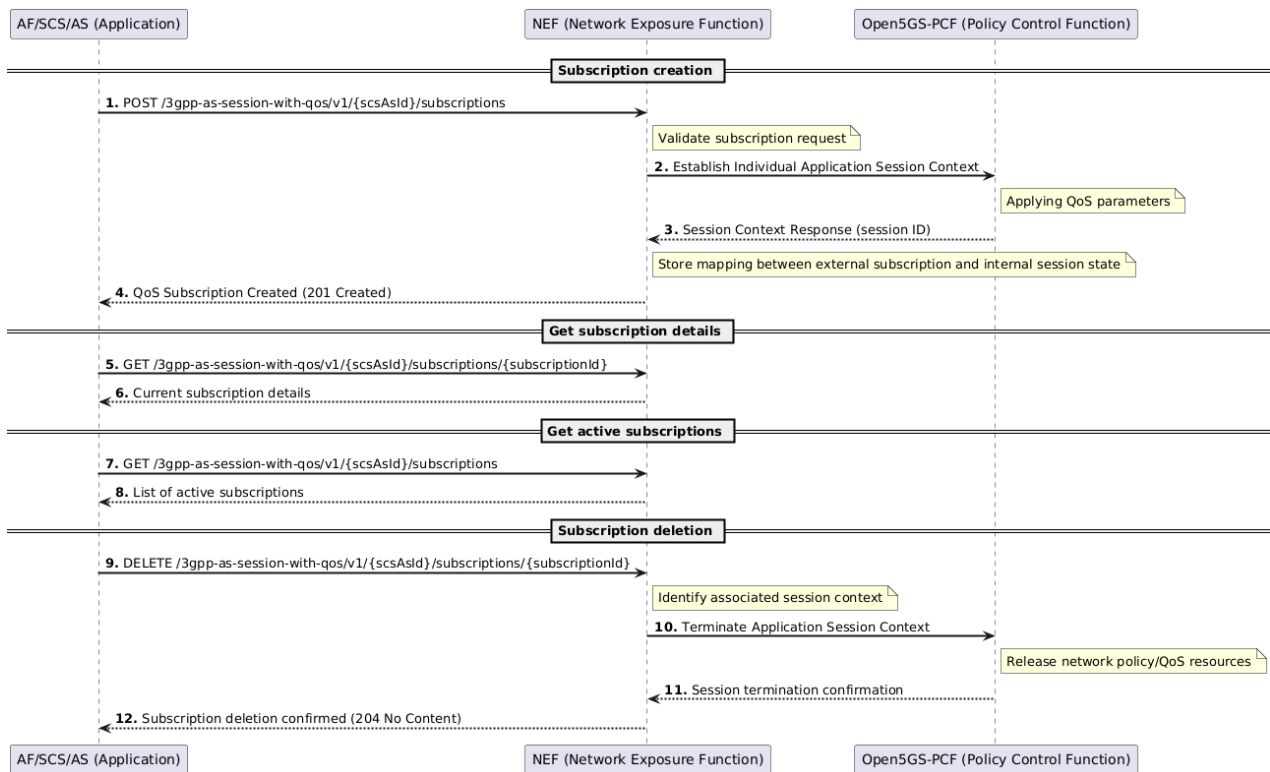


Figure 2 – Sequence Diagram AsSessionWithQoS APIs.

The following explanation corresponds to the sequence diagram shown in Figure 2.

1. AF/SCS/AS sends POST /3gpp-as-session-with-qos/v1/{scsAsId}/subscriptions to the NEF to create a QoS subscription. This request contains a UE address or a list of UE addresses, flow description information that specifies the associated application session and the requested QoS information (e.g. a 5QI value).
2. NEF sends a Policy Authorization Create request to PCF and establishes an Individual Application Session Context on the PCF. Normally, the NEF uses the UE address in step 1 to discover (from Binding Support Function - BSF) the PCF associated with the UE. However, this is not required here since the B5G network contains only one PCF.

At this point, the PCF updates its policies and initiates a Session Management (SM) Policy Association Modification procedure as defined in TS 23.502 clause 4.16.5.2 to notify the Session Management Function (SMF) about the modification of policies. In turn, the SMF initiates the Protocol Data Unit (PDU) Session Modification procedure as defined in TS 23.502 clause 4.3.3 to provide the requested QoS information and flow description to the Radio Access Network (RAN) and User Plane Function (UPF). This enforces the user plane functions to apply the requested QoS information for the associated application session.

3. PCF returns the Session Context Response (session ID) to the NEF.
4. NEF replies to the AF with 201 Created, confirming the QoS subscription.
5. Optionally, the AF may send GET /3gpp-as-session-with-qos/v1/{scsAsId}/subscriptions/{subscriptionId} to retrieve a specific subscription.

6. NEF returns the current subscription details to the AF.
7. Optionally, the AF may send GET /3gpp-as-session-with-qos/v1/{scsAsId}/subscriptions to retrieve the list of all active subscriptions.
8. NEF returns the list of active subscriptions to the AF.
9. If the AF wants to remove an existing individual subscription, AF sends DELETE /3gpp-as-session-with-qos/v1/{scsAsId}/subscriptions/{subscriptionId} to remove this subscription.
10. NEF instructs the PCF to terminate the Application Session Context.
11. PCF returns session termination confirmation to the NEF.
12. NEF responds to the AF with 204 No Content, confirming deletion.

Throughout this flow, the NEF mediates all AF interactions with the PCF, exposing simple create, read and delete operations for managing QoS application sessions.

3.1.4 API endpoints

The *AsSessionWithQoS* API includes 4 RESTful Hypertext Transfer Protocol (HTTP) requests which are described below.

GET	/3gpp-as-session-with-qos/v1/{scsAsId}/subscriptions	Get All Subscriptions Based On Scsas
POST	/3gpp-as-session-with-qos/v1/{scsAsId}/subscriptions	Create Subscription

Figure 3 – Subscription endpoints based on scsAsId.

- POST /3gpp-as-session-with-qos/v1/{scsAsId}/subscriptions: On subscription creation, the NEF derives parameters and establishes an Individual Application Session Context toward the PCF for policy/QoS authorization. If the request is successful, it returns status code 201, otherwise it returns a corresponding error status code.
- GET /3gpp-as-session-with-qos/v1/{scsAsId}/subscriptions: Fetches all subscriptions resources for the specified AF.

GET	/3gpp-as-session-with-qos/v1/{scsAsId}/subscriptions/{subscriptionId}	Get With Scsasid And Subscriptionid
DELETE	/3gpp-as-session-with-qos/v1/{scsAsId}/subscriptions/{subscriptionId}	Delete With Scsasid And Subscriptionid

Figure 4 – Subscription endpoints based on scsAsId and subscriptionId.

- GET /3gpp-as-session-with-qos/v1/{scsAsId}/subscriptions/{subscriptionId}: Retrieves a specific subscription resource by its Universally Unique Identifier (UUID).

- **DELETE /3gpp-as-session-with-qos/v1/{scsAsId}/subscriptions/{subscriptionId}**: Deletes the specified subscription. The NEF terminates the corresponding Individual Application Session Context and revokes the PCF policy/QoS.

3.1.5 Request and response examples

Below is illustrated the example for **POST /3gpp-as-session-with-qos/v1/{scsAsId}/subscriptions** payload.

```
{
  "flowInfo": [
    {
      "flowDescriptions": [
        "permit in ip from 10.45.0.4 to any",
        "permit out ip from any to 10.45.0.4"
      ],
      "flowId": 1
    }
  ],
  "notificationDestination": "https://example.com/callback",
  "qosReference": "qod_2",
  "supportedFeatures": "12",
  "ueIpv4Addr": "10.45.0.4"
}
```

The request body includes the following required parameters:

- **flowInfo**: Describes the IP data flow which requires QoS.
 - **flowDescriptions**: It's a list containing the packet filters of the IP flow. For instance: the filter "permit in ip from 10.45.0.4 to any" allows incoming (uplink) IP packets originating from the UE with the IP address 10.45.0.4 to any destination, while "permit out ip from any to 10.45.0.4" permits outgoing (downlink) IP packets from any source to this UE. The direction "in" refers to uplink IP flows, and the direction "out" refers to downlink IP flows. Please refer to subclause 5.3.8 of 3GPP TS 29.214 for more information.
 - **flowId**: Indicates the identity of the IP flow.
- **notificationDestination**: Contains the Uniform Resource Locator (URL) of the AF to receive the notification bearer level event(s) from the NEF.
- **qosReference**: Identifies pre-defined QoS information.
 - e.g.1 qod_2 = { "marBwDI": "20 Mbps", "marBwUI": "20 Mbps", "mediaType": "AUDIO" }
 - e.g.2 qod_3 = { "marBwDI": "40 Mbps", "marBwUI": "40 Mbps", "mediaType": "VIDEO" }
 - The above examples request downlink and uplink 20 Mbps for AUDIO and 40 Mbps for VIDEO.
- **supportedFeatures**: A string used to indicate the features supported by an API as defined in clause 6.6.2 in 3GPP TS 29.500. The string contains a bitmask indicating supported features in hexadecimal representation. Each character in the string takes a value from "0"

to "9", "a" to "f" or "A" to "F" and represents the support of 4 features as described in table 5.2.2-3. The most significant character, representing the highest-numbered features, appears first in the string, and the character representing features 1 to 4 appears last. The list of features and their numbering (starting with 1) are defined separately for each API. If the string is shorter than required to represent all defined features for an API, any features corresponding to the missing (rightmost) positions in the string are not supported. The features supported by Open5gs are the following:

- OGS_SBI_NPCF_POLICYAUTHORIZATION_IMS_SBI → 5
- OGS_SBI_NPCF_POLICYAUTHORIZATION_SPONSORED_CONNECTIVITY → 2
- The bitmask: 00010010 → 12
- **uelpv4Addr:** The Ipv4 address of the UE.

3.1.6 Error codes & error handling

The status codes and their handling are implemented as specified in TS 29.122, clause 5.2.6. Error responses may include the ProblemDetails data structure to provide additional context and specific error information when encountering application-level failures or validation errors.

3.2 MonitoringEvent API

3.2.1 Short description

The NEF *EventMonitoring* feature enables external AF to monitor specific network events through the NEF. One of the exposed events is *location reporting* which allows AFs to receive notifications about a UE's location updates. This includes reporting based on triggers such as entering or leaving a geographic area, periodic updates, or significant movement thresholds. NEF securely interfaces with core functions like Access and Mobility Management Function (AMF) and Gateway Mobile Location Centre (GMLC) to normalize and expose location data while enforcing authorization and privacy policies. This capability supports advanced use cases like geofencing, asset tracking, and location-based services for vertical applications.

The *EventMonitoring* API will be used as Native API for the ENVELOPE Device Location (Geofencing Subscriptions) API that will be used to support ENVELOPE use cases to subscribe to the location of devices (e.g., when vehicles enter a geofenced area) and trigger other APIs such as QoD as detailed in section 4.

3.2.2 Relation with standards

The *MonitoringEvent* API developed in the project complies with the API defined in clause 5.3 of 3GPP TS 29.122 V17.10.0 specification and supports the monitoring type LOCATION_REPORTING.

3.2.3 Functional overview

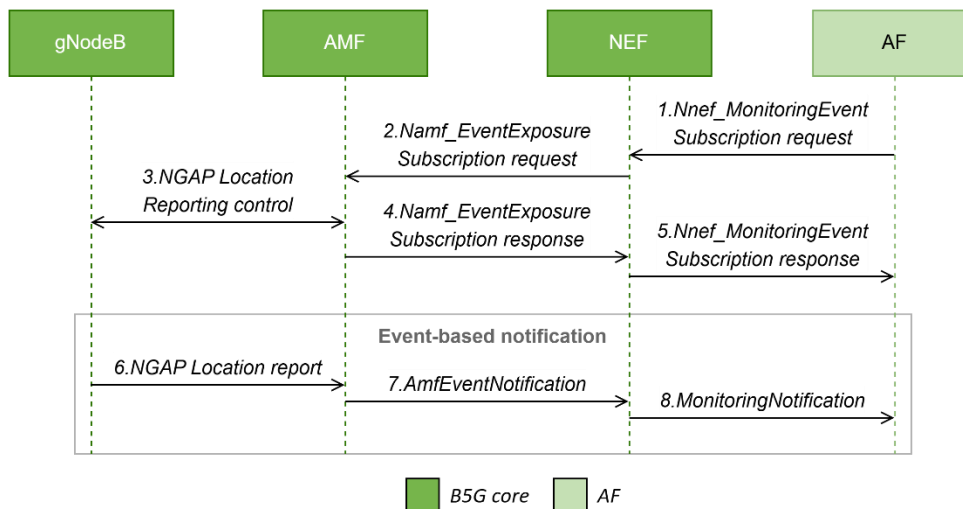


Figure 5 – Sequence diagram for MonitoringEvent API for location reporting.

Figure 5 shows the sequence diagram for MonitoringEvent API for location reporting:

1. The AF sends a request to subscribe to event monitoring information for a UE (i.e., MonitoringEvent) via NEF i.e., to receive location information for a specific UE. The LocationType is the request message set to CURRENT_LOCATION or LAST_KNOWN_LOCATION. The UE is identified by its IPv4 address. The event monitoring periodicity is defined in seconds. Finally, the only location accuracy supported in this implementation is based on cell ID level.
2. The NEF sends a subscription request to the AMF component for receiving the corresponding EventExposure notifications.
3. The AMF sends an NG Application Protocol (NGAP) Location Reporting Control message to the gNodeB to subscribe to changes in radio cell for a particular UE. In case the UE is not in CM_CONNECTED state, a network triggered service request is triggered before sending the NGAP message.
4. AMF confirms the subscription requested by NEF.
5. NEF confirms the subscription requested by the AF.
6. Whenever there is a radio cell change performed by the UE, e.g., due to mobility, the gNodeB sends an NGAP Location Report to the AMF.
7. AMF sends the event notification to the NEF.
8. NEF sends the event notification to the AF.

3.2.4 API endpoints

The endpoints for the *MonitoringEvent* API are described below. Figure 6 shows the Monitoring Event *subscriptions* endpoints.



GET	<code>/[{scsAsId}]/subscriptions</code>	Read all or queried active subscriptions for the SCS/AS.	
POST	<code>/[{scsAsId}]/subscriptions</code>	Creates a new subscription resource for monitoring event notification.	

Figure 6 – MonitoringEvent *subscriptions* endpoints.

- GET `/{{scsAsId}}/subscriptions`: Lists all (or filtered) monitoring-event subscriptions for the given AF (application server) uniquely identified by the `scsAsId` query parameter. It returns a list of `MonitoringEventSubscription` objects.
- POST `/{{scsAsId}}/subscriptions`: Creates a new subscription resource for monitoring event notification for a given AF (e.g., application server) uniquely identified by the `scsAsId` parameter. After a subscription is created, callback `MonitoringNotification` objects are periodically sent to the subscriber notification destination endpoint.

Figure 7 shows the Individual Monitoring Event *subscription* endpoints.





GET	<code>/{{scsAsId}}/subscriptions/{{subscriptionId}}</code>	Read an active subscriptions for the SCS/AS and the subscription Id. 
PUT	<code>/{{scsAsId}}/subscriptions/{{subscriptionId}}</code>	Updates/replaces an existing subscription resource. 
PATCH	<code>/{{scsAsId}}/subscriptions/{{subscriptionId}}</code>	Modifies an existing subscription of monitoring event. 
DELETE	<code>/{{scsAsId}}/subscriptions/{{subscriptionId}}</code>	Deletes an already existing monitoring event subscription. 

Figure 7 – MonitoringEvent subscription-specific endpoints.

- GET `/{{scsAsId}}/subscriptions/{{subscriptionId}}`: Retrieves the details of a specific monitoring-event subscription. Returns the full `MonitoringEventSubscription` resource if it exists.
- PUT `/{{scsAsId}}/subscriptions/{{subscriptionId}}`: Replaces/updates an existing subscription with the provided representation. Returns 200 with the updated resource (or 204 with no body) on success.
- PATCH `/{{scsAsId}}/subscriptions/{{subscriptionId}}`: Partially modifies an existing subscription using JavaScript Object Notation (JSON) Patch (e.g., add/remove specific UEs in a group). Returns 204 when the change is applied.
- DELETE `/{{scsAsId}}/subscriptions/{{subscriptionId}}`: Deletes the subscription; if final event reports arrived, they can be returned in the response. Returns 204 'No Content' or 200 with an array of `MonitoringEventReport`.

3.2.5 Request and response examples

The following examples illustrate the different payloads for subscription creation (request) and corresponding location reporting notifications (response).

An example for POST `/{{scsAsId}}/subscriptions` with `MonitoringEventSubscription` payload for location reporting is given below. The following fields are used:

- **notificationDestination**: The application server Uniform Resource Identifier (URI) where the monitoring event notifications will be delivered to.
- **monitoringType**: Specifies the type of monitoring event to be reported, such as location reporting.
- **repPeriod**: Indicates the reporting period in seconds for periodic notifications of the monitoring event.
- **locationType**: Defines which location information is requested, e.g., current location or last known location.
- **ipv4Addr**: IPv4 address of the UE to be monitored.

```
{
  "notificationDestination": "http://<app-server>/<resource-path>/<event>",
  "monitoringType": "LOCATION_REPORTING",
  "repPeriod": 1,
  "locationType": "CURRENT_LOCATION",
  "ipv4Addr": "10.45.0.2"
}
```

An indicative response to the aforementioned request is provided with *MonitoringNotification* payload below. The following fields are used for location reporting notifications with cell ID:

- **subscription**: Identifier of the monitoring event subscription associated with this notification.
- **monitoringEventReports**: A list of reports containing details about the monitored events of the subscription.
- **monitoringType**: Indicates the type of monitoring event reported, such as location reporting.
- **locationInfo**: Provides location-related information of the UE when the event occurred.
- **ageOfLocationInfo**: The elapsed time in seconds since the location information was obtained.
- **cellId**: Identifier of the serving cell where the UE was located.
- **trackingAreaId**: Identifier of the tracking area associated with the UE's location.
- **plmnId**: Public Land Mobile Network identifier representing the operator's network.

```
{
  "subscription": "AFname",
  "monitoringEventReports": [
    {
      "monitoringType": "LOCATION_REPORTING",
      "locationInfo": {
        "ageOfLocationInfo": 1,
        "cellId": "301",
        "trackingAreaId": "52",
        "plmnId": "20469",
      },
    },
  ]
}
```

3.2.6 Error codes & error handling

The status codes and their handling are implemented as specified in 3GPP TS 29.122 V17.10.0, clauses 5.2.6 and 5.3.5.

3.3 MEC Location API

3.3.1 Short description

The ETSI MEC Location Service is a service of the MEC Platform that is responsible for gathering UE location information and providing this information to external AFs. The location information is exposed by the ETSI MEC Location API. The Location Service can provide the location information of UEs, the distance of UEs from specific points and the list of UEs in a given area.

In this specific implementation of the MEC Location Service, the UEs' positions are sent to the Location Service from a service running on the UEs. This service is named Vehicle Location Service, given that the UEs are On-Board Units (OBUs) as we are considering the CAM application vertical. The Vehicle Location Service retrieves the position from the Global Navigation Satellite System (GNSS) receiver at the UE side and it sends the information to the MEC Location Service. The information is sent using a publish-subscribe (pub/sub) approach, where producers publish messages to a message broker on named topics and any subscribed consumers automatically receive them. The MEC Location API is important for the purposes of the ENVELOPE project, as it enables the injection of CAM-related data into the 5G network and makes this data accessible to end-user applications. The implemented approach also provides accurate positioning information, which is fundamental for several CAM use cases. In addition, the API interacts with the Device Location enabler by exposing the information required for the Devices in Area API.

The MEC Location API offers two endpoints: i) the User Area Subscribe API, and ii) the User Discovery API. The implementation of the MEC Location API has been initiated in a previous project (i.e., 5G-IANA), and it has been extended in ENVELOPE by implementing the User Area Subscribe API as an additional endpoint of the ETSI MEC Location API. This endpoint allows an API consumer to subscribe to receive notifications of UE location events (i.e., UE entering the zone, UE leaving the zone). The endpoint User Discovery API was implemented in the 5G-IANA project⁷. The goal of this endpoint is to provide the precise position of each single UE. This endpoint was defined following the data model and the approach of the other endpoints introduced in the ETSI MEC Location APIs specification.

3.3.2 Relation with standards

The ETSI MEC Location API is defined in the ETSI GS MEC 013 V3.2.1.

3.3.3 Functional overview

⁷ <https://www.5g-iana.eu/>

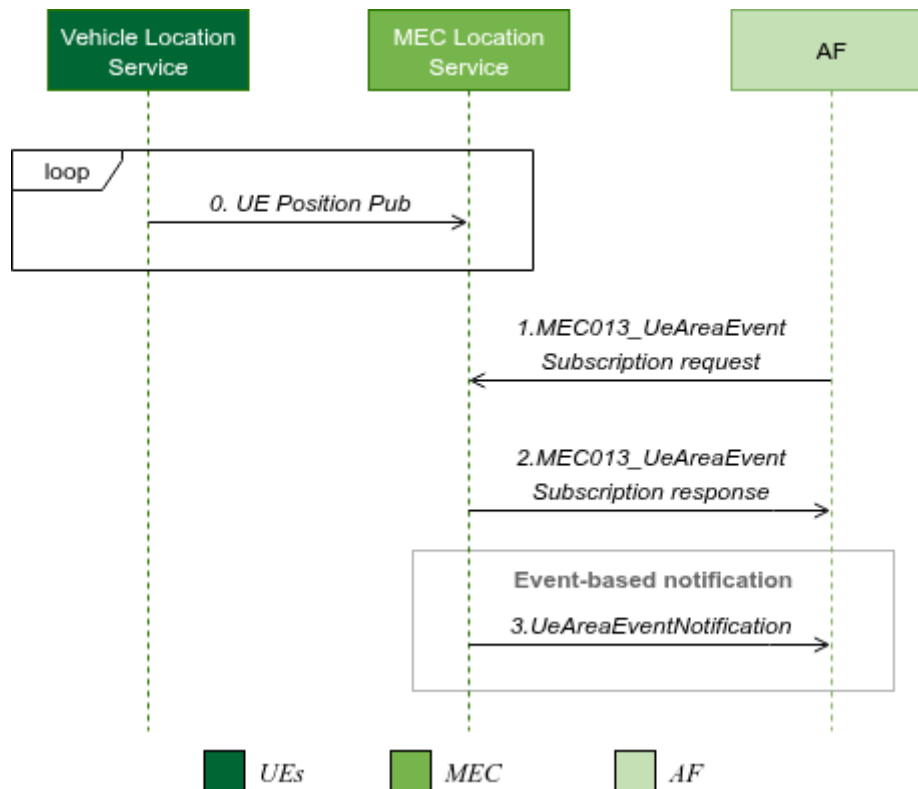


Figure 8 – Sequence diagram for MEC013 – User Area Subscribe API.

Figure 8 shows the sequence diagram for the MEC013 – User Area Subscribe API:

0. The Vehicle Location Service at the UE-side periodically provides position information to the MEC Location Service.
1. The AF subscribes to the User Area Subscribe API to receive notifications about the UE movements in relation to a geographical area.
2. The MEC Location Service returns the response to the subscription request with the identifier of the subscription and the selected area.
3. The MEC Location Service sends notifications to the AF related to UE movement.

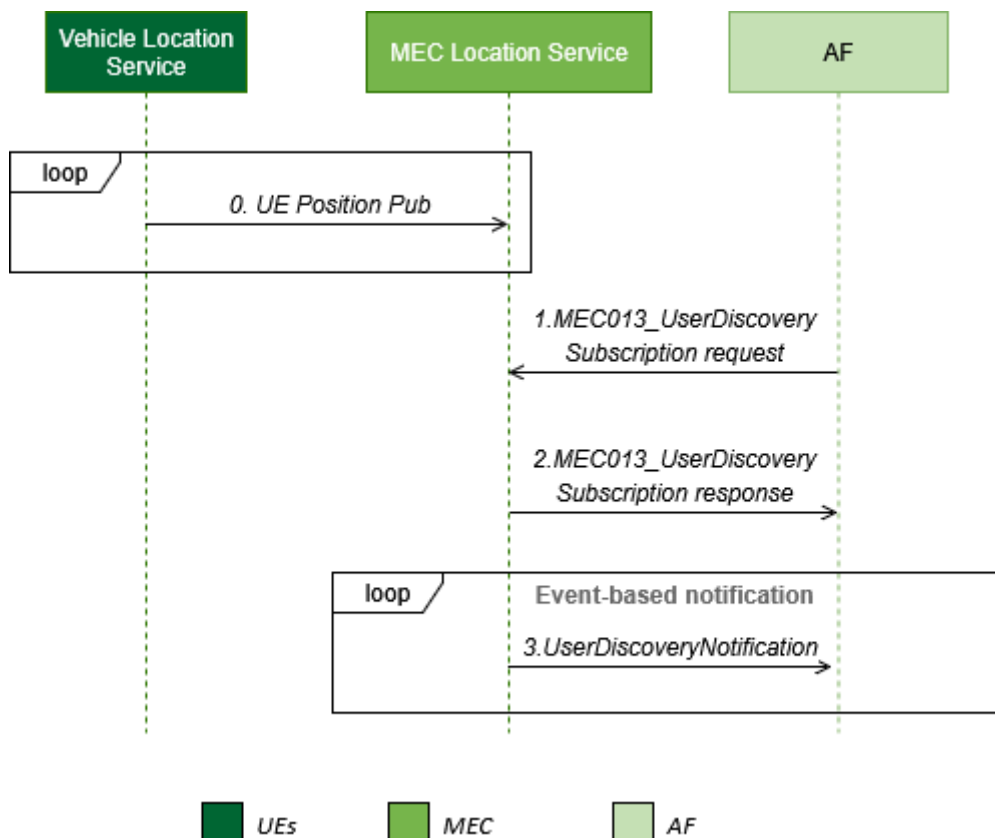


Figure 9 – Sequence diagram for MEC013 – User Discovery Subscribe API.

Figure 9 shows the sequence diagram for the MEC013 – User Discovery API:

0. The Vehicle Location Service at the UE-side periodically provides position information to the MEC Location Service.
1. The AF subscribes to the User Discovery API to receive notifications about the UE position.
2. The MEC Location Service returns the response to the subscription request with the identifier of the subscription.
3. The MEC Location Service sends notifications to the AF related to UE position.

3.3.4 API endpoints

The endpoints for the User Area Subscribe API are described below. Figure 10 shows the corresponding *subscriptions* endpoints.

GET	/subscriptions/area	Retrieves information about the subscriptions for this requestor.	▼
POST	/subscriptions/area	Creates subscription to area notifications.	▼

Figure 10 – User Area Subscribe *subscriptions* endpoints.

- GET /subscriptions/area: Lists all UE Area subscriptions for the given AF. It returns a list of UserAreaSubscription objects.
- POST /subscriptions/area: Creates a new subscription resource for User Area Subscribe notification for a given AF. After a subscription is created, callback UserAreaNotification objects are periodically sent to the subscriber's notification destination endpoint, these callback objects provide information about the events related to UEs in the selected area (e.g., UE entering or leaving the area).

Figure 11 shows the corresponding *subscriptions-specific* endpoints.

GET	/subscriptions/area/{subscriptionId}	Retrieve subscription information	▼
PUT	/subscriptions/area/{subscriptionId}	Updates a subscription information	▼
DELETE	/subscriptions/area/{subscriptionId}	Cancel a subscription	▼

Figure 11 – User Area Subscribe subscription-specific endpoints.

- GET /subscriptions/area/{subscriptionId}: Retrieves the details of a specific User Area subscription. Returns the UserAreaNotification resource if it exists.
- PUT /subscriptions/area/{subscriptionId}: Replaces/updates an existing subscription with the provided representation. Returns 200 with the updated UserAreaNotification resource on success.
- DELETE /subscriptions/area/{subscriptionId}: Deletes the subscription. Returns 204 with no content.

The endpoints for the User Discovery API are described below. Figure 12 shows the corresponding *subscriptions* endpoints.

GET	/subscriptions/discovery	Retrieves information about the subscriptions for this requestor.	▼
POST	/subscriptions/discovery	Creates subscription to UE position notifications.	📄 ↩ ▼

Figure 12 – User Discovery *subscriptions* endpoints.

- GET /subscriptions/discovery: Lists all User Discovery subscriptions for the given AF. It returns a list of UserDiscoverySubscription objects.
- POST /subscriptions/discovery: Creates a new subscription resource for User Discovery Subscribe notification for a given AF. After a subscription is created, callback UserDiscoveryNotification objects are periodically sent to the subscriber notification destination endpoint, these callback objects provide the list of UEs and their position.

Figure 13 shows the corresponding *subscriptions-specific* endpoints.

GET	/subscriptions/discovery/{subscriptionId}	Retrieve subscription information	▼
PUT	/subscriptions/discovery/{subscriptionId}	Updates a subscription information	▼
DELETE	/subscriptions/discovery/{subscriptionId}	Cancel a subscription	▼

Figure 13 – User Discovery subscription-specific endpoints.

- GET /subscriptions/discovery/{subscriptionId}: Retrieves the details of a specific User Discovery subscription. Returns the UserDiscoveryNotification resource if it exists.
- PUT /subscriptions/discovery/{subscriptionId}: Replaces/updates an existing subscription with the provided representation. Returns 200 with the updated UserDiscoveryNotification resource on success.
- DELETE /subscriptions/discovery/{subscriptionId}: Deletes the subscription. Returns 204 with no content.

3.3.5 Request and Response examples

The following is an example of *UserAreaSubscription* payload. In particular, the following fields are present:

- *subscriptionType*: Indicates that the subscription is for the User Area APIs.
- *clientCorrelator*: A correlator that the client can use to tag and track this subscription.
- *callbackReference*: The URL to which notifications will be sent.
- *locationEventCriteria*: The list of location-related events that should trigger notifications.
- *areaDefine*: The geographical area to which the subscription applies.

```
{
  "userAreaSubscription": {
    "subscriptionType": "UserAreaSubscription",
    "clientCorrelator": "3589",
    "callbackReference": "http://callbackurl.com/location_notifications/589347",
    "trackingAccuracy": 5.0,
    "locationEventCriteria": ["ENTERING_AREA_EVENT"],
    "areaDefine": {
      "shape": 1,
      "points": [
        {
          "latitude": 45.58796,
          "longitude": 7.358459
        }
      ],
      "radius": 250
    }
  }
}
```

An indicative response to the aforementioned request is provided in this *UserAreaNotification* payload example. In particular, the following fields are present:

- *notificationType*: Indicates that the notification is from the User Area APIs.
- *address*: the address of the device.

- *userLocationEvent*: The notified event.
- *links*: It provides the hyperlinks related to the resource.

```
{
  "userAreaNotification": {
    "notificationType": "UserAreaNotification",
    "address": "acr:10.0.0.1",
    "userLocationEvent": "ENTERING_AREA_EVENT",
    "_links": {
      Self: "http://mec1.com/mecAPI/location/v3/subscriptions/area/subsc
            ription423"
    }
  }
}
```

The following is an example of *UserDiscoverySubscription* payload. In particular, the following fields are present:

- *subscriptionType*: Indicates that the subscription is for the User Discovery APIs.
- *callbackReference*: The URL to which notifications will be sent.
- *areaDefine*: The geographical area to which the subscription applies in quadtree format.
- *periodicEventInfo*: the interval at which notifications should be sent.

```
{
  "userDiscoverySubscription": {
    "subscriptionType": "UserDiscoverySubscription",
    "callbackReference": "http://location-api-
      consumer:3000/location/v3/consumer/notifications/discovery",
    "areaDefine": {
      "quadTree": [
        "12022301011102.*"
      ],
      "shape": 3
    },
    "periodicEventInfo": {
      "reportingInterval": 5000
    }
  }
}
```

An indicative response to the aforementioned request is provided in this *UserDiscoveryNotification* payload example. In particular, the following fields are present:

- *notificationType*: Indicates that the notification is from the User Discovery APIs.
- *monitoredUsers*: The list of devices with the related position information.
- *areaDefine*: The geographical area to which the subscription applies in quadtree format.
- *Timestamp*: the timestamp at which the notification was generated.
- *links*: It provides the hyperlinks related to the resource.

```
{
  "userDiscoveryNotification": {
    "notificationType": "UserDiscoveryNotification",
    "monitoredUsers": {
```

```

    "user": [
      {
        "address": {
          "publicIP": "192.85.1.23",
          "servicePort": 58453
        },
        "locationInfo": {
          "latitude": 45.064784833,
          "longitude": 7.659256667
        },
        "timestamp": {
          "seconds": 1681303025,
          "nanoSeconds": 650000000
        }
      },
      {
        "address": {
          "publicIP": "192.85.1.32",
          "servicePort": 55234
        },
        "locationInfo": {
          "latitude": 45.064784833,
          "longitude": 7.659256667
        },
        "timestamp": {
          "seconds": 1681303025,
          "nanoSeconds": 650000000
        }
      }
    ],
    "timestamp": {
      "seconds": 1681303025,
      "nanoSeconds": 651991167
    },
    "_links": {
      "subscription": {
        "href":
          "http://mec1.com/mecAPI/location/v3/subscriptions/discovery/d75b36e5-
          2b02-4a0c-8a43-890bb0bef571"
      }
    }
  }
}

```

3.3.6 Error codes & error handling

The status codes and their handling are implemented as specified in ETSI GS MEC 013 V3.2.1 (2024-11) clause 7.16.

3.4 AnalyticsExposure API

3.4.1 Short description

The NEF *AnalyticsExposure* API allows AFs to access network analytics generated by the NWDAF through the NEF such as analytics data related to UE data traffic uplink and downlink throughput

metrics. It enables third-party applications to subscribe to, receive and manage analytics information securely without direct interaction with NWDAF.

The *AnalyticsExposure* API will be used as Native API for the ENVELOPE Performance Metrics API to support ENVELOPE use cases to monitor the uplink and downlink throughput performance as detailed in section 5.

3.4.2 Relation with standards

The *AnalyticsExposure* API developed in the project complies with 3GPP TS 29.522 V17.17.0 and 3GPP OpenAPI specification 3gpp-analyticsexposure v1.1.3.

3.4.3 Functional overview

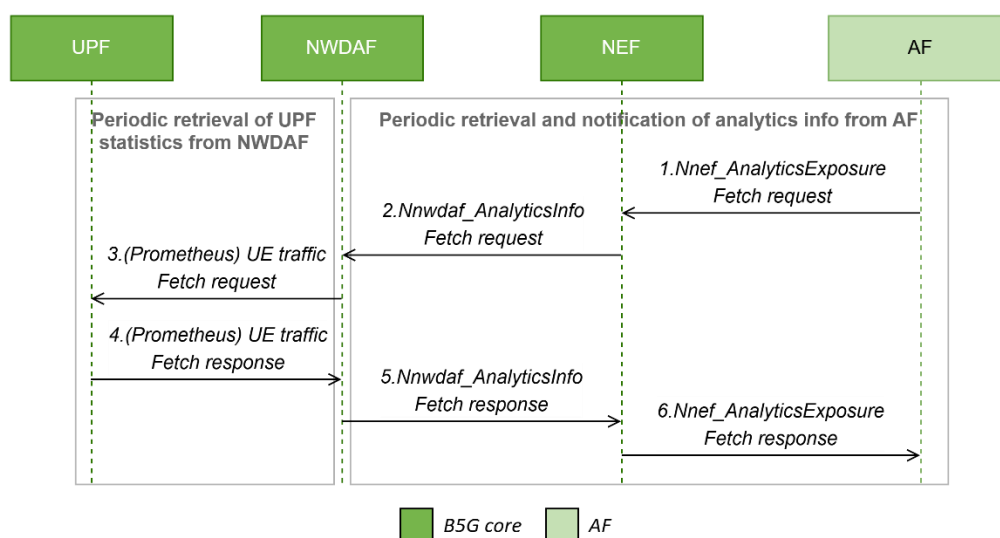


Figure 14 – Sequence diagram for AnalyticsExposure API.

Figure 14 shows the sequence diagram for *AnalyticsExposure* API:

1. The AF performs periodic fetching of analytics information related to the UE's communication (i.e., *UeCommunication*) via NEF, namely, on downlink and uplink volume metrics.
2. The NEF sends a request to the NWDAF component for retrieving the corresponding analytics information.
3. The NWDAF performs a separate periodic fetching of UE performance metrics (i.e., downlink and uplink volume metrics for a UE) from the UPF via a non-3GPP API which relies on Prometheus exporter available at Open5GS for UPF metrics.
4. The UPF replies to the NWDAF with the requested UPF metrics for the specified UE.
5. The NWDAF component replies to the NEF's request with the retrieved performance metrics.
6. The NEF forwards the retrieved information to the AF.

3.4.4 API endpoints

Figure 15 shows the corresponding fetch endpoint.

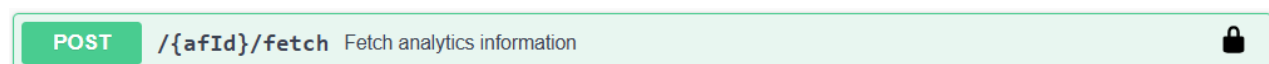


Figure 15 – AnalyticsExposure endpoints.

- **POST /{afld}/fetch:** Fetch analytics for a given AF (application server) uniquely identified by the afld parameter. It returns the AnalyticsData object for successful requests (code 200) containing information related to traffic characteristic data for a requested UE such as uplink and downlink traffic volume; returns 400 for incorrect requests.

3.4.5 Request and response examples

The following examples illustrate the different payloads for periodic fetch (request) and corresponding analytics data notifications (response).

An example for POST /{afld}/fetch with AnalyticsRequest (request) payload is given below. The following fields are used for collecting periodic throughput information for a particular UE via the “UE_COMM” analytics event:

- **analyticsEvent:** Specifies the type of analytics information requested, such as UE communication (“UE_COMM”).
- **tgtUe:** Identifies the target UE for which analytics are requested.
- **gpsi:** A Generic Public Subscription Identifier used to uniquely identify the UE (e.g., Mobile Station International Subscriber Directory Number (MSISDN), external ID, or IP-based identifier).
- **suppFeat:** Indicates the supported features by the AF or NEF for this request, encoded as a feature bitmask string

```
{
  "analyticsEvent": "UE_COMM",
  "tgtUe": {
    "gpsi": "ueIp@envelope",
  },
  "suppFeat": "0"
}
```

An example for *AnalyticsData* (response) payload for the “UE_COMM” event is given below:

- **analyticsEvent:** Indicates the type of analytics event for which data is being provided, e.g., UE communication or mobility.
- **ueCommInfos:** A list of communication-related analytics information for the target UE(s).
- **commDur:** Represents the communication duration in seconds for the reported UE communication session.
- **trafChar:** Contains traffic characteristics observed during the communication session.
- **ulVol:** The total uplink data volume in bytes transmitted by the UE during the session.
- **dlVol:** The total downlink data volume in bytes received by the UE during the session.
- **suppFeat:** Indicates supported features for the analytics data, encoded as a feature bitmask string

```
{
  "analyticsEvent": "UE_COMM",
  "ueCommInfos": [
    {
      "commDur": 1,
      "trafChar": {
        "ulVol": 15204321,
        "dlVol": 385204812
      }
    }
  ],
}
```

```
"suppFeat": "2"  
}
```

3.4.6 Error codes & error handling

The status codes and their handling are implemented as specified in 3GPP TS 29.122 V17.10.0 clause 5.2.6 and 3GPP TS 29.522 V17.17.0 clause 5.6.5.

4 ENVELOPE APIs Design and Implementation

4.1 Quality on Demand (QoD) API

4.1.1 Short description

The QoD API provides a programmable interface that enables developers to request specific service levels, such as guaranteed throughput or consistent low latency, for their applications' data streams. By hiding the complexity of underlying technologies like 4G and 5G networks, the API removes the need for developers to master detailed network configurations.

Within the ENVELOPE framework, QoD APIs are especially significant, as they allow application functions to achieve minimal latency in V2V and V2I communications. This capability improves the ENVELOPE use cases, such as autonomous driving and collision prevention, where even tiny delays of just a few milliseconds can affect safety outcomes.

4.1.2 Relation with standards

The API is based on the Quality on Demand standard API⁸ within the CAMARA Project. The CAMARA QoD project is essentially a family of APIs, which differ as follows: qos-profiles lists available network quality profiles, quality-on-demand enables temporary high-performance network sessions and qod-provisioning supports persistent policies for automatic quality activation in future connections. The QoD ENVELOPE enabler is an implementation of the qos-profiles and quality-on-demand.

4.1.3 Functional overview

The following sequence diagram describes the interaction between the AF layer with the QoD enabler.

⁸ <https://camaraproject.org/quality-on-demand/>

Retrieve available QoS Profiles

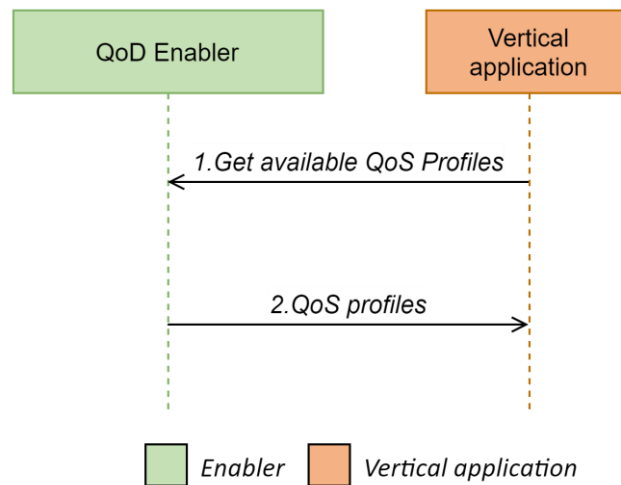


Figure 16 – Sequence Diagram for QoS Retrieve available QoS Profiles request API.

1. The vertical application requests the QoS enabler for the list of available QoS Profiles
2. The QoS enabler provides the list of available QoS profiles that the application is enabled to request

Creation of a QoS session

Initiates a new QoS session. An application requests specific network quality, and the API confirms session establishment with relevant details.

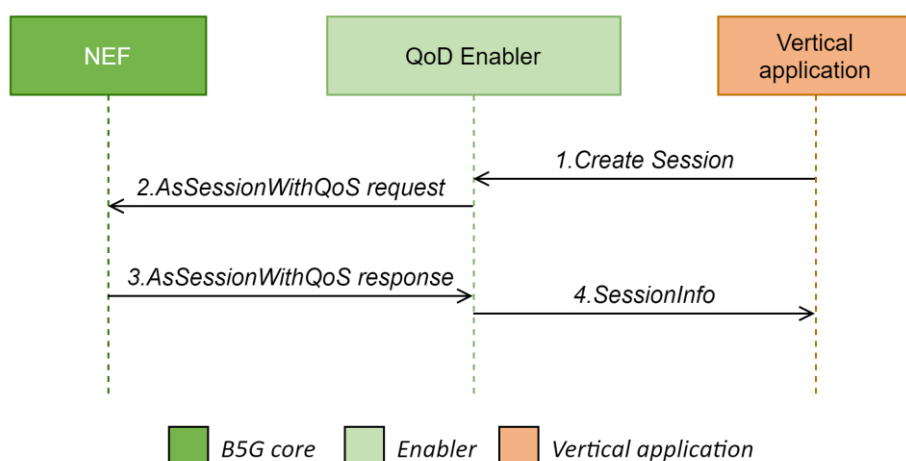


Figure 17 – Sequence Diagram for QoS session request API.

1. AF Consumer requests the creation of a new prioritized data flow (QoS session) to the AF Layer.

2. The QoD enabler uses the NEF to invoke the 3GPP AsSessionWithQoS API, enabling applications to request specific network quality-of-service parameters for their sessions.
3. AF Layer confirms the establishment of the prioritized data flow with session details, enabling enhanced communication.

Get Individual QoS session

Retrieves details of a specific, existing QoS session. Allows applications to check the status and parameters of an active session.

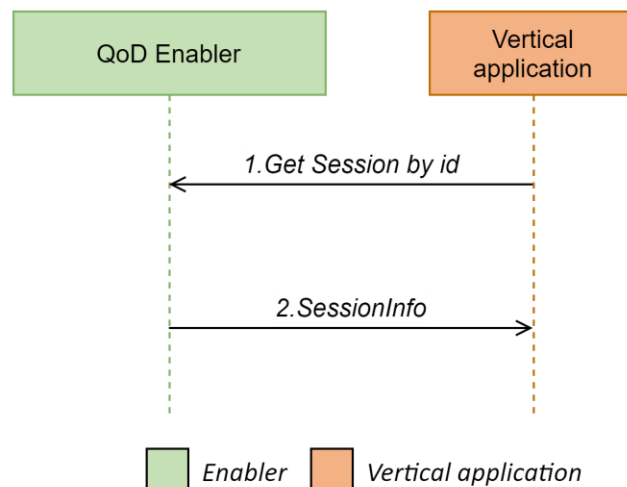


Figure 18 – Sequence Diagram for QoD get individual session info API.

1. AF Consumer queries the AF Layer to retrieve details of a specific, active prioritized data flow.
2. AF Layer provides current information about the requested prioritized data flow's parameters and status for monitoring.

Delete Individual QoS session

Terminates an existing QoS session. This releases reserved network resources when the specific quality is no longer needed.

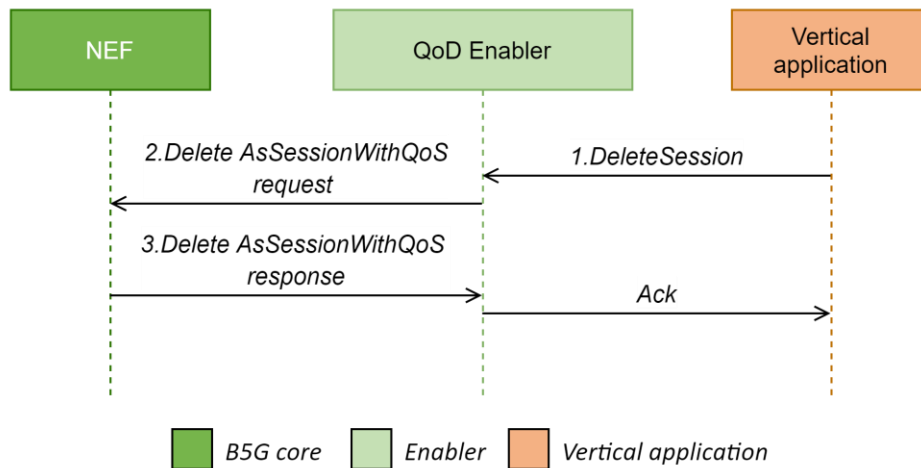


Figure 19 – Sequence Diagram for QoD Delete individual session info API.

1. AF Consumer sends a request to the AF Layer to terminate an existing prioritized data flow.
2. QoD enabler delete the corresponding session resource through the corresponding NEF API.
3. AF Layer confirms the release of network resources previously allocated for that specific prioritized data flow.

Retrieve all QoS sessions

Fetches information on all active QoS sessions for a given consumer. Provides an overview for managing multiple active sessions.

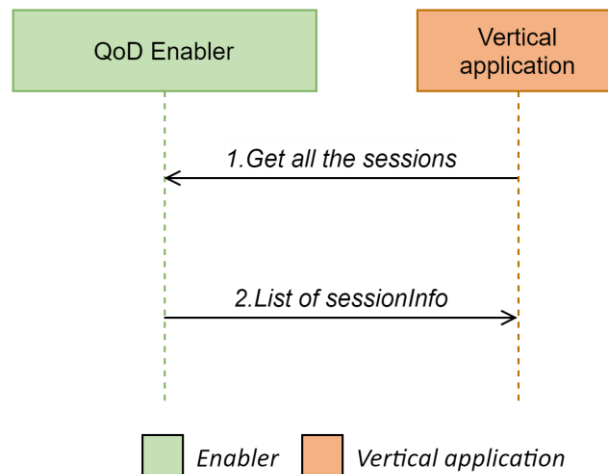


Figure 20 – Sequence Diagram for QoS Retrieve all QoS sessions API.

1. AF Consumer requests from the AF Layer a list of all currently active prioritized data flows for its services.
2. AF Layer provides an overview of all established prioritized data flows to the AF Consumer.

Extension of a QoS session

Extends the duration of an active QoS session. Allows applications to prolong guaranteed QoS without creating a new session.

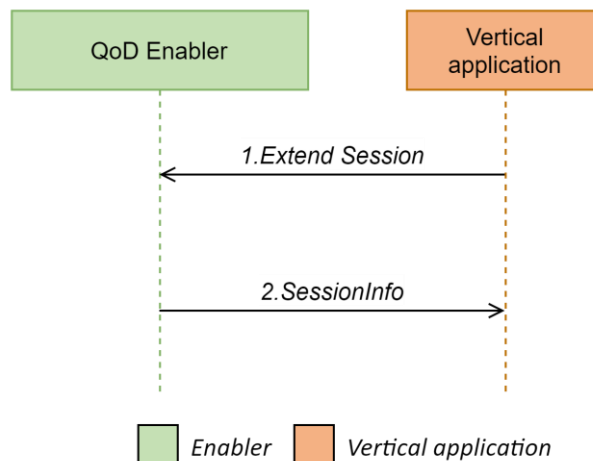


Figure 21 – Sequence Diagram for QoS Extension of a QoS session API.

1. AF Consumer sends a request to the AF Layer to extend the duration of an existing prioritized data flow.
2. AF Layer confirms the extended validity of the prioritized data flow, ensuring continuous low-latency communication.

The QoD enabler consists of the CAMARA QoD API layer combined with a set of Transformation Functions (TFs) specifically adapted to the underlying automotive infrastructure resources exposing the functionalities. The Italian site enabler maps the QoD APIs into the Native API exposed by the HPE Core Network (mainly through the AsSessionWithQoS API from the NEF), while the same enabler is mapped into the same 3GPP API but exposed by the Open5GS open source 5G Core Network implementation, which also supports the 3GPP API. This flexible approach allows each site to tailor the TF to its own network environment.

4.1.4 API endpoints

QoS Profiles <small>Manage QoS Profiles</small>		^
POST	/retrieve-qos-profiles Retrieve QoS profiles	🔒 ▼
GET	/qos-profiles/{name} Get QoS Profile for a given name	📄 🔒 ↩ ▼

QoS Sessions <small>Manage QoS sessions</small>		^
POST	/retrieve-sessions Get QoS session information for a device	▼ 🔒
POST	/sessions Creates a new session	▼ 🔒
DELETE	/sessions/{sessionId} Delete a QoS session	▼ 🔒
GET	/sessions/{sessionId} Get QoS session information	▼ 🔒
POST	/sessions/{sessionId}/extend Extend the duration of an active session	▼ 🔒

Figure 22 – Swagger UI for QoD APIs.

Resource: QoS Profiles

- Resource URI: {apiRoot}/{apiName}/{apiVersion}/retrieve-qos-profiles
- Content type: All request and response bodies use JSON, indicated by setting "application/json".

Resource methods:

- POST – Retrieve the list of QoS profiles. This call returns all the QoS profiles matching the provided device and filter criteria.
- GET – Retrieve the information about a certain QoS profile, given the QoS Profile name.

Resource: QoS Sessions

- Resource URI: {apiRoot}/{apiName}/{apiVersion}/sessions
- Content type: All request and response bodies use JSON, indicated by setting "application/json".

Resource methods:

- POST – Create a QoS session to manage latency and throughput priorities.
 - If the qosStatus in the API response is AVAILABLE and a notification callback was provided, the API consumer will receive both the response and a QOS_STATUS_CHANGED event with qosStatus = AVAILABLE.
 - If the qosStatus is REQUESTED, the client may receive either:
 - A QOS_STATUS_CHANGED event with qosStatus = AVAILABLE once the network confirms the session creation, or
 - A QOS_STATUS_CHANGED event with qosStatus = UNAVAILABLE and statusInfo = NETWORK_TERMINATED if the network fails to establish the session.
 - Additionally, a QOS_STATUS_CHANGED event with qosStatus = UNAVAILABLE will be sent if the network terminates the session before the requested duration ends.

Resource: Individual QoS Session

- Resource URI: {apiRoot}/{apiName}/{apiVersion}/sessions/{sessionId}

Resource methods:

- GET – Retrieve details of a specific QoS session.
- DELETE – Release resources associated with the QoS session.
 - If a notification callback was provided and the session's qosStatus was AVAILABLE, the client will receive both the response and a QOS_STATUS_CHANGED event with:
 - qosStatus = UNAVAILABLE
 - statusInfo = DELETE_REQUESTED
 - No notification is sent if the session status was already UNAVAILABLE.

Resource: Extension of a QoS Session

- Resource URI: {apiRoot}/{apiName}/{apiVersion}/sessions/{sessionId}

Resource methods:

- POST – Extend the duration of an active QoS session (with qosStatus = AVAILABLE).
 - The total session duration (original + extension) cannot exceed the maximum allowed for the QoS profile.
 - If the requested extension goes beyond the allowed limit, the duration will be capped at the maximum.

Example:

- 1) Maximum allowed duration: 50,000 seconds
- 2) Original session duration: 30,000 seconds
- 3) Requested extension: 30,000 seconds
- 4) New total duration: 50,000 seconds (capped at the maximum)

Resource: Retrieve All Sessions

5) Resource URI: {apiRoot}/{apiName}/{apiVersion}/retrieve-sessions

Resource methods:

- 6) POST – Query details of all QoS session resources for a specific device.

4.1.5 Request and response examples

Here is an example for creating a QoS session in QoD:

Request

POST {apiRoot}/{apiName}/{apiVersion}/sessions

Content-Type: application/json

```
{
  "applicationServer": {
    "ipv4Address": "192.168.100.0/24",
    "ipv6Address": "2001:db8:beef::/64"
  },
  "applicationServerPorts": "80,443",
  "device": {
    "ipv4Address": {
      "publicAddress": "10.0.0.50",
      "publicPort": 12345
    },
    "ipv6Address": "2001:db8::1",
    "phoneNumber": "+49123456789"
  },
  "devicePorts": "5000-5100",
  "qosProfile": "LOW_LATENCY",
  "sink": "https://cam-app.example.com/notifications",
  "sinkCredential": "Bearer abc123token",
  "duration": 1800
}
```

Where the following fields are used for the body:

- **applicationServer:** specifies the IP addresses ication server handling the data stream
- **applicationServerPorts:** Lists the TCP/UDP ports on the application server that handle traffic.

- **device:** contains device identification properties to specify the endpoint needing QoS, including:
 - **ipv4Address and publicPort:** IP address and port of the device in IPv4
 - **ipv6Address:** public IPv6 address of the device
 - **PhoneNumber:** telephone number associated with the device
- **devicePorts:** the network ports on the device used for communication.
- **qoSProfile:** the name of the pre-defined QoS profile being requested
- **sink:** the callback endpoint URL where QoS status change events and notifications will be sent to the application.
- **sinkCredential:** authorization token or credential required to authenticate the notification callback, securing event delivery.
- **duration:** Duration in seconds for which the QoS session is requested.

Response

HTTP/1.1 201 Created

Content-Type: application/json

```
{
  "applicationServer": {
    "ipv4Address": "192.168.100.0/24",
    "ipv6Address": "2001:db8:beef::/64"
  },
  "applicationServerPorts": "80,443",
  "device": {
    "ipv4Address": {
      "publicAddress": "10.0.0.50",
      "publicPort": 12345
    },
    "ipv6Address": "2001:db8::1",
    "networkAccessIdentifier": "cam-device@example.com",
    "phoneNumber": "+49123456789"
  },
  "devicePorts": "5000-5100",
  "qoSProfile": "LOW_LATENCY",
  "sink": "https://cam-app.example.com/notifications",
  "sinkCredential": "Bearer abc123token",
  "duration": 1800,
  "expiresAt": "2025-10-17T13:00:00Z",
  "qoSStatus": "REQUESTED",
  "sessionId": "d47a2b4e-2d18-4b3b-97f9-3e7a91287c1f",
  "startedAt": "2025-10-17T12:30:00Z",
  "statusInfo": ""
}
```

Where the following fields (beside the ones that have been already explained above in the request) are returned to the response body:

- **expiresAt:** The timestamp indicating when the QoS session reservation will expire
- **qosStatus:** The current status of the QoS session, such as "REQUESTED" (session creation pending), "AVAILABLE" (session active and confirmed), or "UNAVAILABLE" (session failed or terminated).
- **sessionId:** A unique identifier assigned to the QoS session.
- **startedAt:** The timestamp when the QoS session was started.
- **statusInfo:** Additional information or reason related to the session status, such as "DURATION_EXPIRED" or "NETWORK_TERMINATED," providing context for status changes or failures.

4.1.6 Error codes & error handling

The status codes and their handling are implemented as specified in the openAPI specification file of CAMARA QoD within the qos-profiles and quality-on-demand api specification.

4.2 Location Retrieval API

4.2.1 Short description

The Location Retrieval API enables vertical applications retrieving the last known location of a mobile device as detected by the mobile network operator. By exposing standardized interfaces, it enables external applications to query device location information without requiring direct interaction with complex 3GPP network functions. The API returns the detected location in geometric forms like polygons depending on the precision available from the network. Some use cases include vehicle location tracking for emergency response coordination (enabling rapid identification of nearby vehicles during accidents or hazardous events), geofenced service activation (triggering QoD requests when vehicles enter specific operational areas) and inter-Public Land Mobile Network (PLMN) handover support (maintaining location awareness during cross-domain mobility for seamless service continuity). This API is important for the purposes of ENVELOPE project as it particularly allows CAM applications to obtain precise vehicle positioning data without complex 3GPP network integration, enabling location-aware service orchestration across trial sites.

4.2.2 Relation with standards

This API complies with the CAMARA Device Location (Location Retrieval) specification version v0.4.0⁹.

4.2.3 Functional overview

The following sequence flow diagram describes the interaction between the CAMARA Device Location API for Location Retrieval feature, when requested for a vAPP, and the NEF

⁹ https://github.com/camaraproject/DeviceLocation/blob/r2.2/code/API_definitions/location-retrieval.yaml

MonitoringEvent API within a 5G communication environment.

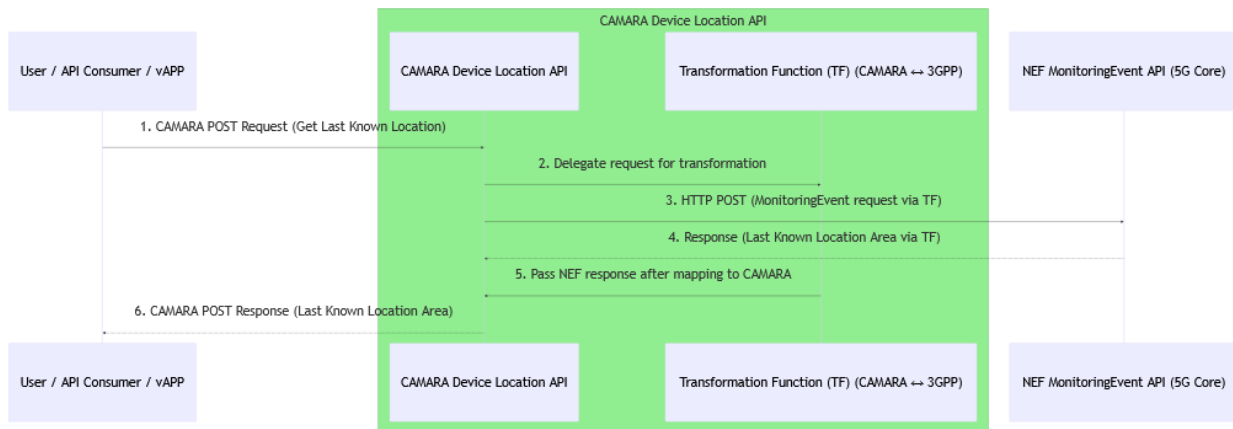


Figure 23 – Message flow for Device location (Location Retrieval) API.

1. The process is initiated by an API consumer(vAPP) issuing an HTTP POST request to the CAMARA Device Location (Location Retrieval) API in order to obtain the last known location area of a target device.
2. In the Location Retrieval API, a transformation function is used internally to handle protocol and payload adaptation. This function translates the CAMARA-compliant request into the required 3GPP-compliant MonitoringEvent request body.
3. CAMARA API issues an HTTP POST request to the NEF MonitoringEvent API with the translated 3GPP compliant Request Body.
4. The NEF processes this request by querying its subscriber database and, if the device is found, returns the corresponding last known location area.
5. The response is then passed back into the CAMARA sidecar, where the 3GPP payload is transformed into the standardized CAMARA response schema.
6. Finally, the CAMARA API delivers the resulting HTTP response to the consumer.

4.2.4 API endpoints

The endpoint for the Device Location for Location Retrieval feature is described below. Figure 24 illustrates the API's endpoint in the context of the northbound interface that a vertical app will use.

Location retrieval

POST /location-retrieval/v0.4/retrieve Retrieve Location

Retrieve the area where a certain user device is localized.

Figure 24 – Location Retrieval API endpoint.

POST /location-retrieval/v0.4/retrieve: Retrieve the last known location of a specific device defined in the request's body. After the validation of the request body, the usage of a transformation function is applied to send a 3GPP compliant POST request to the 5G via NEF's *MonitoringEvent* API. Returns status code 200, for successful location retrieval or 400, 401, 403, 404, 422 for bad requests, unauthorized, forbidden requests or not found related devices and if validation fails respectively.

Figure 25 showcases the operation in the 5G system, via NEF's *MonitoringEvent* API, in the context

of a southbound interface through the use of the transformation function described above. That API is not part of the CAMARA API, thus only the internal usage of it in the scope of CAMARA is described.

MonitoringEvent API Subscription level POST Operation

POST	/3gpp-monitoring-event/v1/{scsAsId}/subscriptions	Create Subscription
Creates a new subscription resource for monitoring event notification		

Figure 25 – MonitoringEvent API endpoint as southbound communication.

POST /3gpp-monitoring-event/v1/{scsAsId}/subscriptions: Fetch the last known location of a specific device that has been previously recorded in NEF's Monitoring Event database and sends it back to the CAMARA Location Retrieval API. Returns status code 200.

4.2.5 Request and response examples

Example for HTTP POST Request for Device Location API payload of Location Retrieval feature:

```
{
  "device": {
    "phoneNumber": "+306912345678"
  }
}
```

The above example defines the device's phone number that will be used, as a request body, to perform a HTTP POST request to fetch the last known location of the requested device, if any.

The following payload shows the response body of the HTTP Response for location retrieval of a requested device.

```
{
  "lastLocationTime": "2025-09-07T10:40:52Z",
  "area": {
    "areaType": "POLYGON",
    "boundary": [
      {
        "latitude": 50.735851,
        "longitude": 7.10066
      },
      {
        "latitude": 50.735851,
        "longitude": 7.10066
      },
      {
        "latitude": 50.735851,
        "longitude": 7.10066
      }
    ]
  }
}
```

Which is detailed as follows:

- **LastLocationTime:** Last date and time when the device was localized. It must follow RFC 3339 and must have time zone.
- **Area:** The area is defined as a Polygon by using list of Points that corresponds to geographical coordinates described by (latitude, longitude).

4.2.6 Error codes & error handling

The status codes and their handling are implemented as specified in the openAPI specification file of CAMARA Device Location API for Location Retrieval feature.

4.3 Device Location (Geofencing Subscriptions) API

4.3.1 Short description

The Device Location (Geofencing Subscriptions) API allows vertical applications to subscribe to geofenced event-based notifications related to the location of specified devices, i.e., when a device enters or leaves a certain geographical area. This feature may be used in various use cases requiring tracking of movement of assets or users. For example, logistics companies may use this API to receive real-time alerts whenever a shipment enters or exits a designated area. Also, a service provider may trigger other APIs (Quality-on-Demand, or Edge Cloud) based on location updates of a particular user.

The Device Location (Geofencing Subscriptions) API will be used to support ENVELOPE use cases to subscribe to the location of devices (e.g., when vehicles enter a geofenced area) and trigger other APIs such as Quality-on-Demand as detailed in section 5.

4.3.2 Relation with standards

This specification complies with the CAMARA Device Location (Geofencing Subscriptions) specification version v0.3.0.

4.3.3 Functional overview

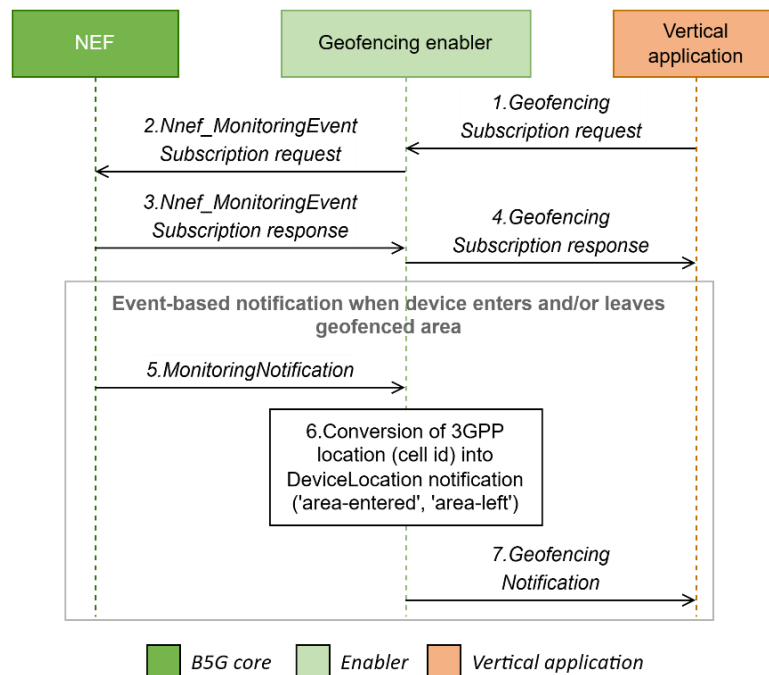


Figure 26 – Sequence diagram the Device Location (Geofencing) API.

The sequence diagram for the Device Location (Geofencing) API is shown in Figure 26.

1. The vertical application subscribes to event-based Device Location (geofencing) notifications for a specified UE by specifying the geofenced area and events to which the UE would like to subscribe (i.e., 'area-entered', 'area-left').
2. The Geofencing enabler subscribes to event monitoring information (i.e., *MonitoringEvent*) via NEF, namely, on location information (e.g., cell id).
3. NEF confirms the subscription requested by the Geofencing enabler.
4. The Geofencing enabler confirms the subscription request to the vertical application.
5. Whenever there is a new monitoring event, NEF sends the event notification to the Geofencing enabler.
6. The Geofencing enabler converts the 3GPP location notification (cell ID) into Device Location notification ('area-entered', 'area-left').
7. Finally, the Device Location notification is sent to the vertical application whenever the subscription conditions are met (i.e., a certain device entered/left a geographical area).

4.3.4 API endpoints

The endpoints for the Device Location (Geofencing subscriptions) API are described below. Figure 27 shows the corresponding *subscriptions* endpoints.

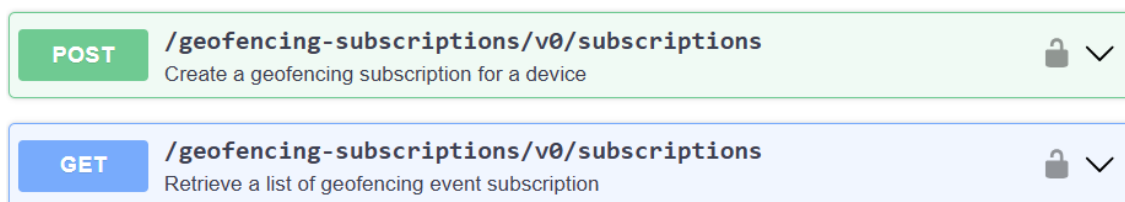


Figure 27 – Geofencing subscription endpoints.

- POST /geofencing-subscriptions/subscriptions: Creates a subscription for a device to receive notifications when a device enters or exits a specified geographical area. After a subscription is created, callback CloudEvent objects are periodically sent to the subscriber notification destination endpoint. Returns 201 for successfully created subscriptions; returns 400 for incorrect requests.
- GET /geofencing-subscriptions/subscriptions: Retrieves a list of geofencing event subscription(s). Returns 200 for successful requests; returns 400 for incorrect requests.

Figure 28 shows the corresponding *subscriptions-specific* endpoints.

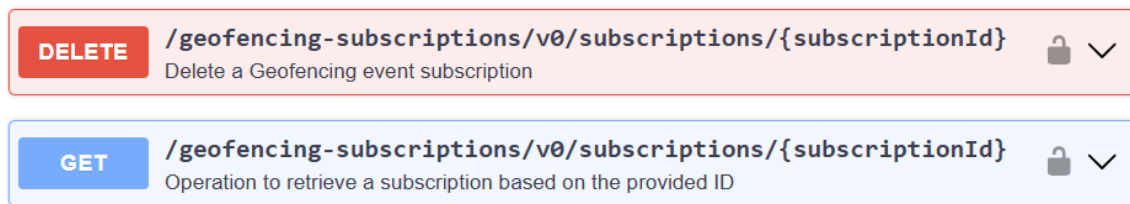


Figure 28 – Geofencing subscription-specific endpoints.

- DELETE /geofencing-subscriptions/subscriptions/{subscriptionId}: Deletes a Device Location (Geofencing) subscription. Returns 200 for successful requests; returns 400 for incorrect requests.
- GET /geofencing-subscriptions/subscriptions/{subscriptionId}: Retrieves a Device Location (Geofencing) subscription. Returns 200 for successful requests; returns 400 for incorrect requests.

4.3.5 Request and response examples

The following examples illustrate different payloads for subscription creation (request) and corresponding geofencing notifications (response).

An example for POST /geofencing-subscriptions/subscriptions with SubscriptionRequest payload for geofencing notification is given below. The following fields are used:

- **protocol**: Specifies the communication protocol used for delivering notifications (e.g., HTTP).
- **sink**: The callback URL where geofencing event notifications will be sent.
- **types**: A list of event types the client subscribes to, following the CloudEvents specification (e.g., area-entered, area-left).
- **config**: Contains configuration details for the subscription, including device and area information.
- **subscriptionDetail**: Holds the main subscription parameters such as device identifier and geofence definition.
- **device**: Identifies the device for which geofencing events are monitored (e.g., by phone number).
- **phoneNumber**: The MSISDN of the device being monitored.
- **area**: Defines the geofenced area where events will be triggered.
- **areaType**: Specifies the shape of the geofence (e.g., CIRCLE).
- **center**: Provides the geographic coordinates (latitude and longitude) of the geofence center.
- **latitude**: Latitude of the geofence centre in decimal degrees.
- **longitude**: Longitude of the geofence centre in decimal degrees.

- **radius:** Radius of the geofence in meters.
- **initialEvent:** Boolean indicating whether an initial event should be sent immediately upon subscription creation.
- **subscriptionMaxEvents:** Maximum number of events to be sent before the subscription ends.
- **subscriptionExpireTime:** The timestamp (ISO 8601 format) when the subscription will expire.

```
{
  "protocol": "HTTP",
  "sink": "https://notificationSendServer12.supertelco.com",
  "types": [
    "org.camaraproject.geofencing-subscriptions.v0.area-entered"
  ],
  "config": {
    "subscriptionDetail": {
      "device": {
        "phoneNumber": "+12345678912"
      },
      "area": {
        "areaType": "CIRCLE",
        "center": {
          "latitude": 50.735851,
          "longitude": 7.10066
        },
        "radius": 2000
      }
    },
    "initialEvent": true,
    "subscriptionMaxEvents": 10,
    "subscriptionExpireTime": "2024-03-22T05:40:58.469Z"
  }
}
```

An example for callback notification with *CloudEvent* payload for geofencing event is given below. The following fields are used:

- **id:** A unique identifier for the event, ensuring it can be distinguished from other events.
- **source:** The URI identifying the origin of the event, typically the notification service endpoint.
- **type:** The event type, indicating the nature of the occurrence (e.g., *org.camaraproject.geofencing-subscriptions.v0.area-entered*).
- **specversion:** The version of the CloudEvents specification used (e.g., 1.0).
- **datacontenttype:** The media type of the event data (e.g., *application/json*).
- **time:** The timestamp when the event was generated, in RFC 3339 format.
- **data:** The event-specific payload containing details of the geofencing event.
- **subscriptionId:** Identifier of the geofencing subscription that triggered the event.
- **device:** Information about the device involved in the event.
- **phoneNumber:** The phone number associated with the device.
- **area:** The geofenced area definition that triggered the event.
- **areaType:** The shape type of the geofence (e.g., *CIRCLE*).
- **center:** The geographical centre of the geofence.
- **latitude:** Latitude coordinate of the centre point.
- **longitude:** Longitude coordinate of the centre point.
- **radius:** The radius of the geofence in meters.

```
{
  "id": "123655",
  "source": "https://notificationSendServer12.supertelco.com",
  "type": "org.camaraproject.geofencing-subscriptions.v0.area-entered",
  "specversion": "1.0",
  "datacontenttype": "application/json",
  "time": "2023-03-22T05:40:23.682Z",
  "data": {
    "subscriptionId": 987654321,
    "device": {
      "phoneNumber": 123456789
    },
    "area": {
      "areaType": "CIRCLE",
      "center": {
        "latitude": 50.735851,
        "longitude": 7.10066
      },
      "radius": 2000
    }
  }
}
```

4.3.6 Error codes & error handling

The status codes and their handling are implemented as specified in CAMARA Device Location (Geofencing Subscriptions) specification version v0.3.0.

4.4 Devices In Area API

4.4.1 Short description

The Device Location – Devices in Area API allows vertical applications to query the list of UEs that are in a geographical area in a given past time interval or to subscribe to geofenced event-based notifications, i.e., when a device enters or leaves a certain geographical area. The main difference with respect to the Device Location – Geofencing Subscriptions is that no information about the identity of the UEs must be provided in the subscription requests. This API is important for the purposes of the ENVELOPE project, as it makes information from the CAM vertical available to experimenters. Position data and related geo-events are fundamental for many CAM applications, since device mobility is a key feature of the CAM vertical. In the context of It-UC1, the Devices in Area API is exploited to identify devices (i.e., vehicles) located near the accident area, which can then be contacted to request and collect sensor data. In It-UC2, the same API is leveraged to detect when vehicles enter a geographical area where an HD map update is available, so that the relevant vehicles can be identified and provided with the updated map.

4.4.2 Relation with standards

The specification of Devices in Area API follows the same approach (i.e., data model) as other CAMARA Device Location APIs, but it is not among the already defined specifications of CAMARA Device Location APIs.

4.4.3 Functional overview

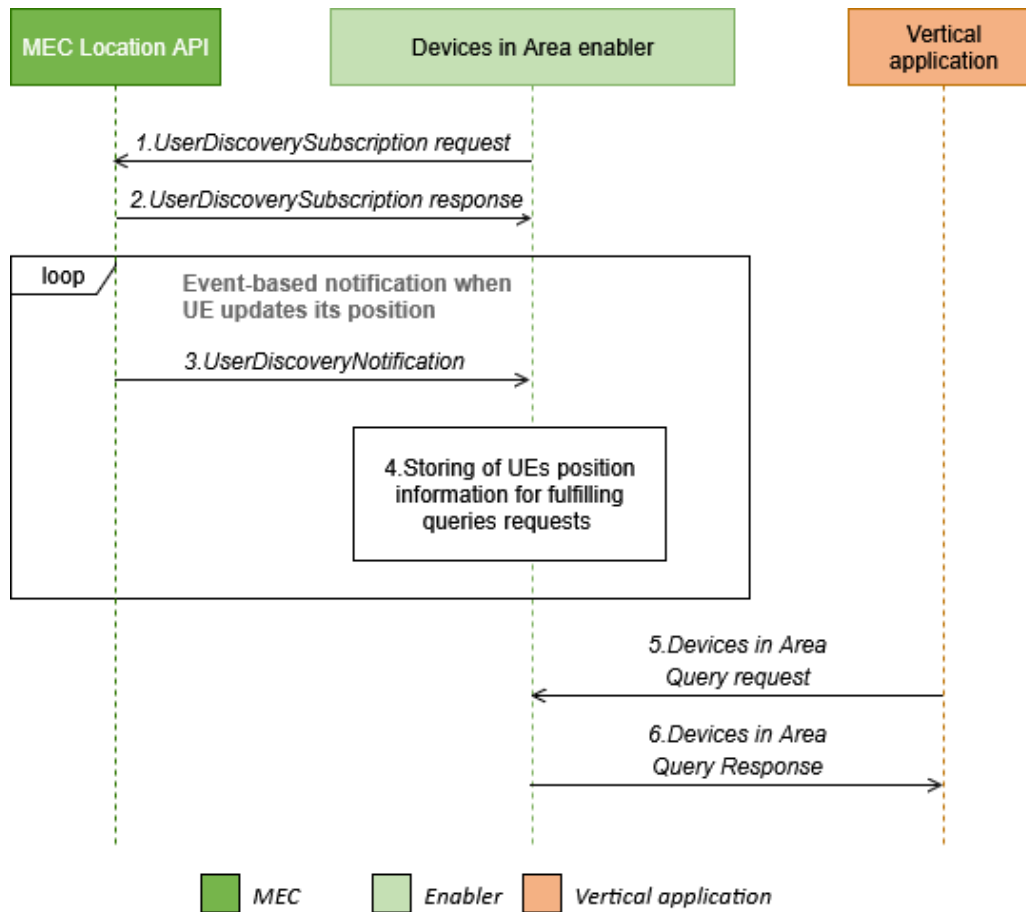


Figure 29 – Sequence diagram of the Devices in Area Query API.

The sequence diagram for the Device Location - Devices in Area API query is shown in Figure 29.

1. The Devices in Area enabler subscribe to MEC Location API – User Discovery for receiving updates of UE positions.
2. The MEC Location API confirms the subscription request.
3. The MEC Location API sends notifications about UEs position to the Devices in Area enabler.
4. The Devices in Area enabler stores UEs position for further processing when queries are performed.
5. The vertical application performs a query to the Device Location – Devices in Area query specifying a geographical area and the time interval to be considered.
6. The Devices in Area enabler replies to the vertical application.

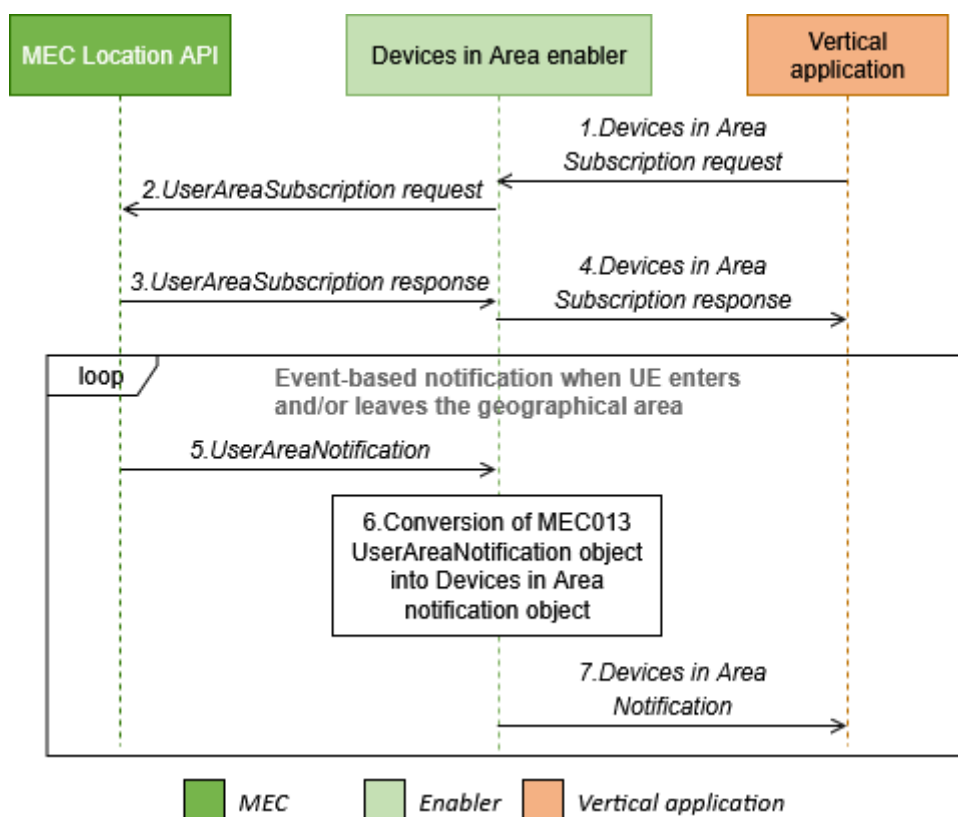


Figure 30 – Sequence diagram the Devices in Area Subscription API.

The sequence diagram for the Device Location - Devices in Area API subscription is shown in Figure 30.

1. The vertical application subscribes to event-based Device Location – Devices in Area notifications specifying a geographical area and events to be reported (i.e., 'area-entered', 'area-left').
2. The Devices in Area enabler subscribes to user area information (i.e., UserAreaSubscribe) via MEC Location API.
3. MEC Location API confirms the subscription requested by the Devices in Area enabler.
4. The Devices in Area enabler confirms the subscription request to the vertical application.
5. Whenever there is a new UE related event, MEC Location API sends the event notification to the Devices in Area enabler.
6. The Devices in Area enabler converts the UserAreaNotification object to Devices in Area notification object.
7. Finally, the Devices in Area notification is sent to the vertical application whenever the subscription conditions are met (i.e., a certain device entered/left a geographical area).

4.4.4 API endpoints

The endpoint for the Device Location - Devices in Area API query is described below. Figure 31 shows the corresponding *query* endpoint.



Figure 31 – Devices in Area query endpoint.

- POST `/devices-in-area/queries/devices`: Performs a query to receive a list of devices in a given geographical area for a past time interval. Upon success of the query, a response with a list of all the devices present in the area is returned.

The endpoints for the Device Location - Devices in Area API are described below. Figure 32 shows the corresponding *subscriptions* endpoints.

POST	<code>/devices-in-area/subscriptions</code>	Create a devices in area subscription	🔒 ▼
GET	<code>/devices-in-area/subscriptions</code>	Retrieve a list of Devices in Area event subscription	🔒 ▼

Figure 32 – Devices in Area subscription endpoints.

- POST `/devices-in-area/subscriptions`: Creates a subscription to receive notifications when devices enter or exit a given geographical area. After a subscription is created, callback CloudEvent objects are periodically sent to the subscriber notification destination endpoint. Returns 201 for successfully created subscriptions; returns 400 for incorrect requests.
- GET `/devices-in-area/subscriptions`: Retrieves a list of geofencing event subscription(s). Returns 200 for successful requests; returns 400 for incorrect requests.

Figure 33 shows the corresponding *subscriptions-specific* endpoints.

GET	<code>/devices-in-area/subscriptions/{subscriptionId}</code>	Operation to retrieve a subscription based on the provided ID	🔒 ▼
DELETE	<code>/devices-in-area/subscriptions/{subscriptionId}</code>	Delete a Devices in Area event subscription	🔒 ▼

Figure 33 – Devices in Area subscription-specific endpoints.

- DELETE `/devices-in-area/subscriptions/{subscriptionId}`: Deletes a Device Location - Devices in Area subscription. Returns 200 for successful requests; returns 400 for incorrect requests.
- GET `/devices-in-area/subscriptions/{subscriptionId}`: Retrieves a Device Location - Devices in Area subscription. Returns 200 for successful requests; returns 400 for incorrect requests.

4.4.5 Request and response examples

The following is an example of Devices in Area query payload. In particular, the following fields are present:

- **area**: The geographical area to which the query applies.
- **fromTime**: The start timestamp from which information should be retrieved.
- **toTime**: The end timestamp up to which information should be retrieved.

```
{
  "area": {
    "areaType": "CIRCLE",
    "radius": 5000,
    "center": {
      "latitude": 45.023646317,
      "longitude": 7.667854067
    }
  }
}
```

```

    }
  },
  "fromTime": 1747919167762,
  "toTime": 1748436146528
}

```

The following is an example of a response payload to the Devices in Area query. The response payload is a JSON array containing the following fields:

- **ipv4Address**: the identifier of the device in the area in the selected time window

```

[
  {
    "ipv4Address": {
      "publicAddress": "60.195.79.145",
      "publicPort": 1234
    }
  },
  {
    "ipv4Address": {
      "publicAddress": "164.128.151.124",
      "publicPort": 5678
    }
  }
]

```

The following is an example of Devices in Area subscription payload. In particular, the following fields are present:

- **protocol**: it indicates the delivery protocol.
- **sink**: The URL to which notifications will be sent.
- **subscriptionDetail**: it contains the details of the subscription.
 - **area**: it defines the area of the subscription.

```

{
  "protocol": "HTTP",
  "sink": "https://devices-in-area-consumer:3000/notifications",
  "config": {
    "subscriptionDetail": {
      "area": {
        "areaType": "CIRCLE",
        "center": {
          "latitude": 45.236745,
          "longitude": 7.854762
        },
        "radius": 250
      }
    }
  }
}

```

The following is an example of *CloudEvent* payload that is the notification sent for a Devices in Area subscription. In particular, the following fields are present:

- **id**: it represents the unique identifier of this event.

- **source**: it provides the identity of the entity responsible for the generation of the event.
- **type**: it indicates the type of the notified event.
- **specVersion**: it specifies the version of the specification.
- **datacontenttype**: it indicates the encoding of the payload.
- **time**: it indicates the timestamp at which the event happened.
- **data**: It contains the payload of the event:
 - **subscriptionId**: it is the unique identifier of the subscription.
 - **device**: it specifies the device involved in the event.
 - **area**: it defines the geographical area of the subscription.

```
{
  "id": "2356",
  "source": "https://devices-in-area-apis:5000/subscriptions",
  "type": "org.camaraproject.geofencing-subscriptions.v0.area-left",
  "specversion": "1.0",
  "datacontenttype": "application/json",
  "time": "2025-09-18T07:40:00.000Z",
  "data": {
    "subscriptionId": 254896,
    "device": {
      "publicAddress": 130.192.22.36,
      "publicPort": 8000
    },
    "area": {
      "areaType": "CIRCLE",
      "center": {
        "latitude": 45.236745,
        "longitude": 7.854762
      },
      "radius": 250
    }
  }
}
```

4.4.6 Error codes & error handling

The status codes and their handling are implemented as specified in specification document of the Devices in Area API available on the ENVELOPE website¹⁰.

4.5 Performance Metrics API

4.5.1 Short description

The Performance Metrics API is a new API developed in the ENVELOPE project to expose periodic network performance metrics to vertical applications. This API allows vertical applications to monitor the current performance of a particular user/device. Currently, uplink and downlink

¹⁰ <https://envelope-project.eu/wp-content/uploads/2025/06/API-specification-Device-Location-Devices-in-area-OC-1-20250602.pdf>

throughput metric values are supported. In future releases, additional metrics may be included such as the total aggregated network throughput and latency.

The Performance Metrics API will be used to support ENVELOPE use cases to monitor the current achieved uplink and downlink throughput performance as detailed in section 5.

4.5.2 Relation with standards

This is a newly proposed API. At the moment of writing, a new CAMARA project in draft state called *Session Insights*¹¹ is being defined, which might cover a similar purpose. Monitoring and alignment with the *Session Insights* CAMARA project will be considered for the remainder of the project.

4.5.3 Functional overview

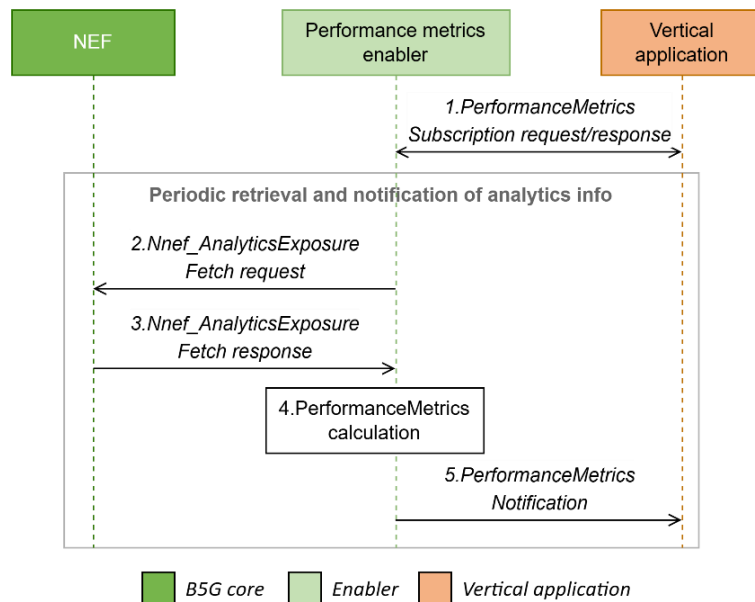


Figure 34 – Message flow for the Performance Metrics API.

The sequence diagram for the Performance Metrics API is shown in Figure 34.

1. The vertical application subscribes to periodic Performance Metrics notifications for a specified UE by giving a list of performance metrics such as uplink/downlink rates (throughput). The Performance metrics enabler confirms the subscription creation request.
2. The Performance metrics enabler performs periodic fetching of analytics information related to the UE's communication (i.e., *UeCommunication*) via NEF, namely, on downlink and uplink volume metrics.
3. The NEF returns the retrieved information back to the Performance metrics enabler for each fetch request.

¹¹ <https://camaraproject.org/session-insights/>

4. The Performance metrics enabler aggregates the retrieved information containing downlink and uplink volume metrics and estimates the corresponding downlink and uplink data rates (throughput) for the specified UE.
5. Finally, a *MetricsReport* object is sent in the form of a periodic notification to the specified vertical application server endpoint.

4.5.4 API endpoints

The endpoints for the Performance Metrics API are described below. Figure 35 shows the corresponding *subscriptions* endpoints.

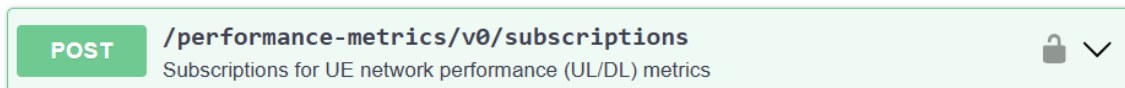


Figure 35 – Performance Metrics subscription endpoints.

- **POST /performance-metrics/subscriptions:** Creates a subscription to receive periodic UE's network performance (UL/DL) metrics. After a subscription is created, callback *MetricsReport* objects are periodically sent to the subscriber notification destination endpoint. Returns 201 for successful created subscriptions; returns 400 for incorrect requests.

Figure 36 shows the corresponding *subscriptions-specific* endpoints.

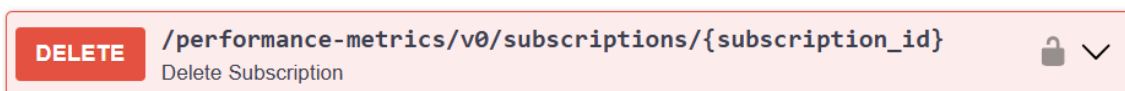


Figure 36 – Performance Metrics subscription-specific endpoints.

- **DELETE /performance-metrics/subscriptions/{subscription_id}:** Deletes a performance metrics subscription. Returns 204 for successfully created subscriptions; returns 404 for subscription not found.

4.5.5 Request and response examples

The following examples illustrate the different payloads for subscription creation (request) and corresponding performance metrics notifications (response).

An example for POST /performance-metrics/subscriptions with CreateSubscription payload for performance metrics notification is given below. The following fields are used:

- **webhook:** Contains information for delivering notifications to the subscriber via a webhook.
- **notificationUrl:** The HTTPs endpoint where performance metric notifications will be sent.
- **notificationToken:** A secret token used for authenticating notifications sent to the webhook endpoint.
- **uelp:** The IP address of the UE for which performance metrics are requested.
- **metricsSelection:** A list of performance metrics to monitor, such as *uplink_rate* and *downlink_rate*.
- **interval:** The reporting interval in seconds for sending performance metric updates.
- **expiresAt:** The timestamp (in RFC 3339 format) indicating when the subscription will expire.

```
{
  "webhook": {
    "notificationUrl": "https://host.envelope.com:8001",
    "notificationToken": "secret"
  },
  "ueIp": "10.45.0.3",
  "metricsSelection": ["uplink_rate", "downlink_rate"],
  "interval": 2,
  "expiresAt": "2025-1-22T05:40:58.469Z"
}
```

An example for callback notification with *MetricsReport* payload for performance metrics notification is given below. The following fields are used:

- **status**: Indicates the result of the metrics report generation or delivery (e.g., Ok for success).
- **timestamp**: The date and time when the metrics were measured or reported, in ISO 8601 format.
- **subscription_id**: Identifier of the subscription associated with this metrics report.
- **ul_rate**: The measured uplink data rate (throughput) in bits per second.
- **dl_rate**: The measured downlink data rate (throughput) in bits per second.

```
{
  "status": "Ok",
  "timestamp": "2025-10-27T08:49:08.837554",
  "subscription_id": "12345678",
  "ul_rate": 1234,
  "dl_rate": 4321
}
```

4.5.6 Error codes & error handling

The status codes and their handling are implemented as specified in the ENVELOPE Performance Metrics API¹².

¹²<https://envelope-project.eu/wp-content/uploads/2025/06/API-specification-Performance-Metrics-OC-1-20250602.pdf>

5 Integration of APIs into the trial sites

5.1 Italian Site - Enabler Mapping and Specific Implementations

The overall architecture of the Italian trial site is illustrated in Figure 37. In this site, two use cases are defined. Depending on the use case, the functionalities offered by the cloud and edge applications illustrated in the figure differ. The common aspect is that the Cloud Application is responsible for monitoring notifications and reacting to these notifications by requesting to the QoS APIs new network QoS profiles for the vehicles that need to share sensor data. Furthermore, the Cloud Application exploits the Devices in Area APIs for getting information about the vehicles in the relevant area of the event (i.e., accident or map update area).

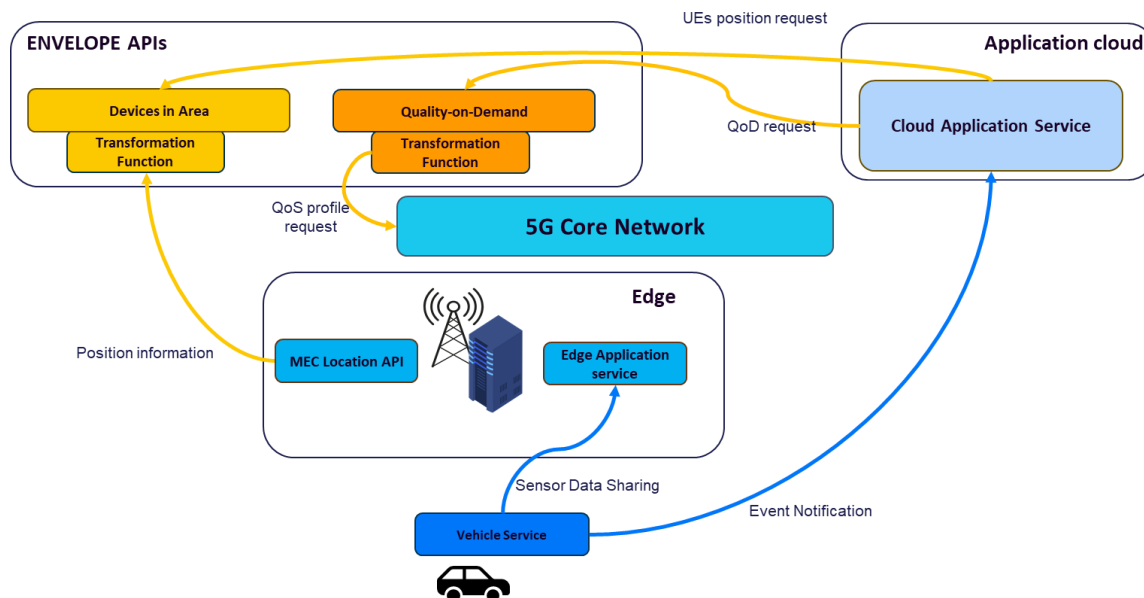


Figure 37 – Overall system architecture in the Italian site.

Below, we highlight the use of the different ENVELOPE APIs presented in this document and their roles in each use case:

It-UC1: Advanced In-Service Reporting for Automated Driving Vehicles:

- The Cloud Accident Reporting Service, after the reception of the accident detection notification, contacts the Devices in Area API to get the list of vehicles in the accident area. Based on the received information, it requests new network QoS profiles via the QoS API to guarantee that the involved vehicles can send the sensor data to the application running at the edge server.
- **ENVELOPE APIs used:** Devices in Area, Quality-on-Demand (QoS).

It-UC2: Dynamic Collaborative Mapping for Automated Driving

- The Cloud Mapping Service, after the reception of the notification of a map update, requests a new network QoS profile via the QoS API to guarantee that the relevant vehicle can send

the sensor data to the application running at the edge server. Furthermore, it contacts the Devices in Area API to get the list of vehicles incoming to the updated map area and, based on the received information, it requests new network QoS profiles via the QoD API to guarantee that the involved vehicles can receive the map updates.

- **ENVELOPE APIs used:** Devices in Area, Quality-on-Demand (QoD).

Finally, in Table 3 below we summarise how the mapping of ENVELOPE API and Native API is defined for each enabler used in the Italian site:

Table 3 – Native and ENVELOPE APIs defined for each enabler used in the Italian site.

Enabler	ENVELOPE API	Native API
Quality-on-Demand (QoD)	CAMARA Quality-on-Demand (QoD)	3GPP AsSessionWithQoS
Device Location – Devices in Area	Devices in Area (based on CAMARA Device Location)	MEC013 – Location API

5.2 Dutch Site - Enabler Mapping and Specific Implementations

The overall architecture of the Dutch site showing the integration of the DT, V2X service and ENVELOPE APIs is illustrated in Figure 38. The system includes a combination of hardware and software installed both in vehicles and back-end edge infrastructure. The 5G-enabled vehicle connects through an OBU with a V2X software stack, while the V2X Application Server in the edge environment manages low-latency API access and message routing. Data from vehicle sensors is processed by an in-vehicle Siemens computer and transmitted to the DT Service using Message Queuing Telemetry Transport (MQTT) via the V2X Application Server, supporting the different DT use cases. The V2X service is responsible for consuming the following ENVELOPE APIs: QoD, Performance metrics, and Device Location.

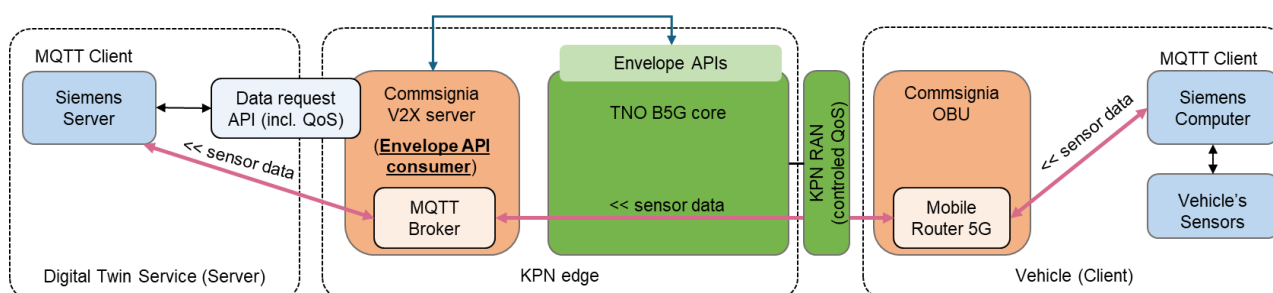


Figure 38 – Overall system architecture in the Dutch site.

In the Dutch site, three DT use cases are defined. Below, we highlight the use of the different ENVELOPE APIs presented in this document and their roles in each use case:

Dt-UC3: Periodic vehicle data collection for improving DT:

- The V2X service requests a network QoS profile via the QoD API to guarantee that the periodic data collection from the fleet of vehicles will be successful and predictable as required by the DT service. The Performance metrics API is consumed by the V2X server to monitor the current achieved uplink and downlink throughput performance.

- **ENVELOPE APIs used:** Quality-on-Demand (QoD), Performance Metrics.

Dt-UC4: Vehicle testing with mixed reality:

- The V2X service requests a network QoS profile via the QoD API to guarantee that the network sessions between the DT service and test vehicle are not disturbed. The Performance metrics API is consumed by the V2X server to monitor the current achieved uplink and downlink throughput performance.
- **ENVELOPE APIs used:** Quality-on-Demand (QoD), Performance Metrics.

Dt-UC5: Tele-operated driving aided by DT:

- The V2X service requests a network QoS profile via the QoD API to guarantee that the network sessions between the DT service and selected road user devices located near the autonomous vehicle with failed sensor are not disturbed. In addition, the Device Location API will enable the V2X service to subscribe to the location of devices (e.g., within a geofenced area) and trigger QoS adjustments only at the relevant location near the vehicle being tele-operated. Finally, the Performance metrics API is consumed by the V2X server to monitor the current achieved uplink and downlink throughput performance.
- **ENVELOPE APIs used:** Quality-on-Demand (QoD), Performance Metrics, Device Location (Geofencing).

Finally, in Table 4 below we summarize how the mapping of ENVELOPE API and Native API is defined for each enabler used in the Dutch site:

Table 4 – Native and ENVELOPE APIs defined for each enabler used in the Dutch site.

Enabler	ENVELOPE API	Native API
Quality-on-Demand (QoD)	CAMARA Quality-on-Demand (QoD)	3GPP AsSessionWithQoS
Device Location (Geofencing)	CAMARA Device Location (Geofencing Subscriptions)	3GPP MonitoringEvent
Performance Metrics	Performance Metrics	3GPP AnalyticsExposure

5.3 Greek Site - Enabler Mapping and Specific Implementations

The overall architecture of the Greek trial site is illustrated in Figure 39. The Greek site demonstrates the ENVELOPE platform's advanced capabilities for inter-PLMN mobility and service continuity through its focus on MEC service handover between two mobile networks (OTE's and NCSRDS's). The Greek use case (UC6) implementation addresses critical challenges in maintaining real-time situation awareness during cross-domain transitions, showcasing B5G innovations that ensure seamless service delivery across heterogeneous network environments.

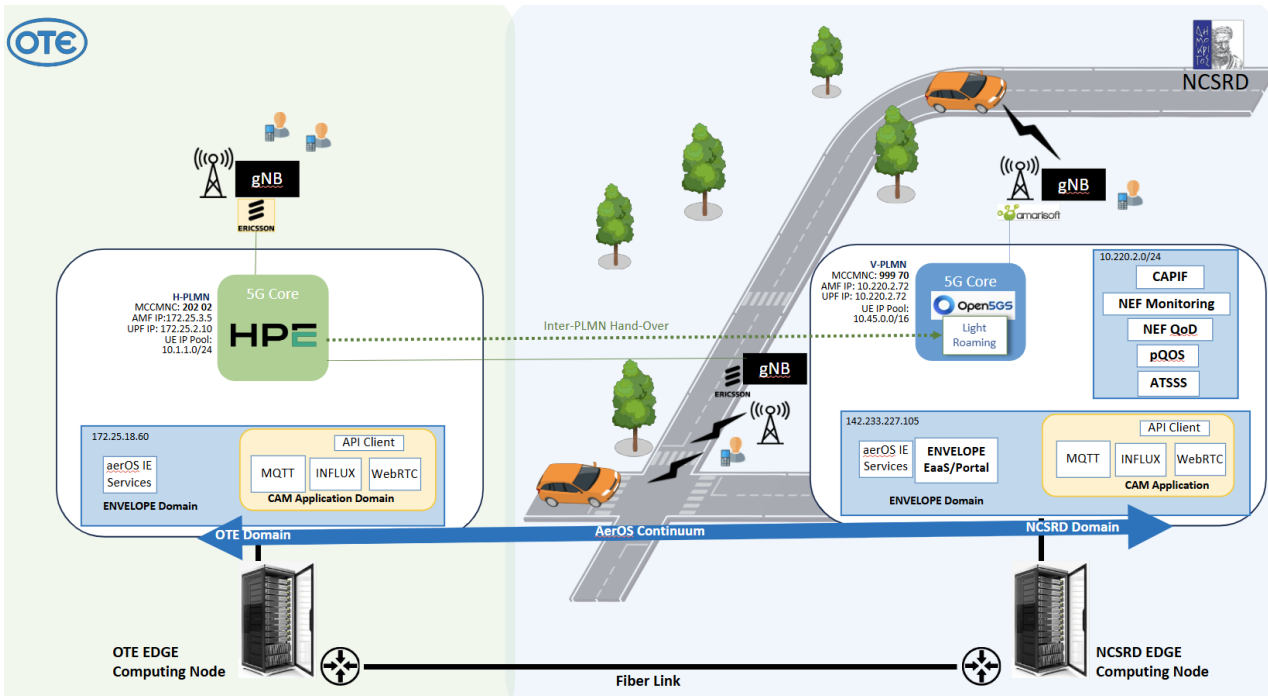


Figure 39 – Overall system architecture in the Greek site.

Below, we highlight the use of the different ENVELOPE APIs presented in this document and their roles in the Greek use case:

Gr-UC6: Data Sharing for Real-Time Situation Awareness with Inter-PLMN Handover:

- The Greek use case exploits QoD API to request enhanced network resources in visited PLMNs when hazardous events are detected, guaranteeing priority treatment for emergency notifications across operator domains. It also exploits location-aware services through exposing Location Retrieval APIs to track vehicle positions during cross-domain mobility, enabling location-specific alerts and seamless service handover triggers based on geographical proximity to PLMN boundaries.
- **ENVELOPE APIs used:** Location Retrieval, Quality-on-Demand (QoD).

Finally, below we summarise how the mapping of ENVELOPE API and Native API is defined for each enabler used in the Greek site:

Table 5 – Native and ENVELOPE APIs defined for each enabler used in the Greek site.

Enabler	ENVELOPE API	Native API
Quality on Demand (QoD)	CAMARA Quality-on-Demand QoD	3GPP AsSessionWithQoS
Device Location	CAMARA Location Retrieval	3GPP MonitoringEvent

6 Next Steps

The next steps particularly related to implementation and integration of APIs include the work performed across WP3 and WP4 to complete platform integration and validation across the three trial sites. The immediate next phase focuses on finalizing the service-oriented capabilities outlined for Deliverable 3.3 (to be delivered by month 24), led by UDE, which will document the remaining innovation developments including pQoS enabler, ATSSS multi-connectivity implementation, MEC handover functionality and aerOS edge continuum management.

The current deliverable represents the first of two parts of APIs implementation activities, where the D3.3 builds upon to serve as the second part of APIs implementation activities. Concurrently, WP4 activities will exploit those APIs for performing trial site integration and pre-evaluation testing. The Deliverable 4.1 (to be delivered by month 27), led by ICCS, will document the trial sites architecture, the development of data collection tools and internet service repository. Finally, Deliverable 4.2 (to be delivered by month 30), led by TNO, will document the integration results and pre-evaluation testing outcomes, serving as the final validation milestone before large-scale demonstrations. Deliverable 4.2 will consolidate inputs from all three trial sites, demonstrating successful API integration, enabler functionality and use case readiness to ensure successful large-scale trials and third-party experimentation through the ENVELOPE platform capabilities developed across both work packages.

7 Conclusions

This deliverable documents the implementation of three fundamental API categories that establish essential network programmability building blocks for B5G systems. QoS APIs combine the Native 3GPP *AsSessionWithQoS* API for standards-compliant network interaction with the CAMARA-compliant QoD API, enabling vertical applications to request guaranteed throughput and consistent latency for safety-critical automotive scenarios without requiring deep 5G technical expertise. Device Location API provides location-aware services through multiple complementary interfaces. These include the Native 3GPP *MonitoringEvent* API for cell-ID based reporting, ETSI MEC Location APIs for precise positioning and northbound CAMARA-compliant interfaces such as Location Retrieval for basic position queries, Device Location Geofencing Subscriptions for area-based alerts, and the innovative "Devices in Area" API specifically designed for identifying vehicles within geographical regions during emergencies or collaborative mapping scenarios. Moreover, Performance Metrics APIs transform complex Network Data Analytics Function information through 3GPP *AnalyticsExposure* APIs into user-friendly interfaces, enabling applications to access real-time network analytics including uplink/downlink measurements and communication patterns essential for DT applications and data-driven resource allocation decisions across all ENVELOPE use cases.

Those APIs are the foundational layer for ENVELOPE's key achievement which is to deliver a modular enabler architecture that effectively transforms complex southbound Native APIs from 5G core networks and edge infrastructure into simplified northbound ENVELOPE APIs. The target is to eliminate the need for vertical applications to master detailed telecommunications specifications while ensuring predictable network performance and programmability.

The segregation of APIs between foundational network exposure capabilities in D3.2 and advanced service-oriented functionalities planned for D3.3 demonstrates essential lessons learned about logical architectural progression, where basic connectivity, monitoring and location awareness must be established before implementing predictive analytics, cross-domain service migration and multi-connectivity features. However, implementation challenges have revealed several limitations. Specifically, there are dependencies on specific 5G core implementations that may limit deployment flexibility across different operator environments, partial inter-PLMN functionality due to technical constraints in supporting equipment compatibility, particularly regarding IMS integration for real smartphone environments and the need for enhanced pQoS capabilities to support per-UE granularity across comprehensive RAN, Edge, Core, and Cloud domains.

Future work will focus on completing the API ecosystem through D3.3, which will introduce advanced capabilities including pQoS enabler, ATSSS multi-connectivity implementation, MEC handover functionality and aerOS edge continuum management, to address current limitations through enhanced online training capabilities for predictive AI/ML models, implement robust MEC federation support and establish seamless service migration across edge domains.