ETSI GR MEC-DEC 063 V4.1.1 (2025-08)

GROUP REPORT

Multi-access Edge Computing (MEC); ESTIMED Analysis and selection for oneM2M and MEC implementations

Disclaimer

The present document has been produced and approved by the Multi-access Edge Computing (MEC) ETSI Industry Specification Group (ISG) and represents the views of those members who participated in this ISG.
It does not necessarily represent the views of the entire ETSI membership.

Reference
DGR/MEC-DEC063EstimedImp

Keywords
M2M, MEC, oneM2M

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI IPR online database.

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™**, **LTE™** and **5G™** logo are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

Foreword

This Group Report (GR) has been produced by ETSI Industry Specification Group (ISG) Multi-access Edge Computing (MEC).

Modal verbs terminology

In the present document "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

1 Scope

The present document provides a summary of the oneM2M and MEC implementations to support standardized IoT deployments in MEC environments. It will include a list of the use case selection criteria and assessment of implementations for use in the PoCs. It will include an evaluation of the existing features that are needed as well as an assessment of development effort required to implement any identified potential new features. Finally, it will indicate the selected implementations and the reasons for their selection.

2 References

2.1 Normative references

Normative references are not applicable in the present document.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long-term validity.

The following referenced documents may be useful in implementing an ETSI deliverable or add to the reader's understanding, but are not required for conformance to the present document.

[i.1] ETSI MEC Overview Standardization update on Multi-access Edge Computing.

[i.2] ETSI MECwiki - MEC Sandbox.

[i.3] ETSI TS 129 222: "LTE; 5G; Common API Framework for 3GPP Northbound APIs (3GPP TS 29.222)".

[i.4] ETSI GS MEC 010-2: "Multi-access Edge Computing (MEC); MEC Management; Part 2: Application lifecycle, rules and requirements management".

[i.5] ETSI GS MEC 011: "Multi-access Edge Computing (MEC); Edge Platform Application Enablement".

[i.6] ETSI GS MEC 012: "Multi-access Edge Computing (MEC); Radio Network Information API".

[i.7] ETSI GS MEC 013: "Multi-access Edge Computing (MEC); Location API".

[i.8] ETSI GS MEC 015: "Multi-Access Edge Computing (MEC); Traffic Management APIs".

[i.9] ETSI GS MEC 021: "Multi-access Edge Computing (MEC); Application Mobility Service API".

[i.10] ETSI GS MEC 028: "Multi-access Edge Computing (MEC); WLAN Access Information API".

[i.11] ETSI GS MEC 029: "Multi-access Edge Computing (MEC); Fixed Access Information API".

[i.12] ETSI GS MEC 033: "Multi-access Edge Computing (MEC); IoT API".

[i.13] ETSI GS MEC 030: "Multi-access Edge Computing (MEC); V2X Information Services API".

[i.14] ETSI GS MEC 040: "Multi-access Edge Computing (MEC); Federation enablement APIs".

[i.15] ETSI GS MEC 046: "Multi-access Edge Computing (MEC); Sensor-sharing API".

[i.16] ETSI White Paper No. #59: "Enabling Multi-access Edge Computing in Internet-of-Things: how to deploy ETSI MEC and oneM2M".

[i.17] ETSI TS 118 123: "oneM2M; SDT based Information Model and Mapping for Vertical Industries (oneM2M TS-0023)".

[i.18] ETSI TS 118 119: "oneM2M; Abstract Test Suite and Implementation eXtra Information for Test (oneM2M TS-0019).

[i.19] ETSI GS MEC 003: "Multi-access Edge Computing (MEC); Framework and Reference Architecture".

3 Definition of terms, symbols and abbreviations

3.1 Terms

Void.

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

ACP	Access Control Policy
AE	Application Entity
AI	Artificial Intelligence
API	Application Programming Interface
ARM	Advanced RISC machine
ASN-CSE	Application Services Node - Common Services Entity
BSD	Berkley Software Distribution
CAPIF	Common API Framework
CBOR	Concise Binary Object Representation
CoAP	Constrained Application Protocol
CORS	Cross-Origin Resource Sharing
CRUD	Create Retrieve Update and Delete
CSE	Common Services Entity
CSF	Common Service Function
CSR	Certificate Signing Request
DTLS	Datagram Transport Layer Security
HTTP	Hypertext Transfer Protocol
ID	IDentifier
IMEI	International Mobile Equipment Identification
IN-CSE	Infrastructure Node - Common Services Entity
IoT	Internet of Thing
IT	Information Technology
JSON	JavaScript Object Notation
JSON-LD	JavaScript Object Notation for Linked Data
MEC	Multi-access Edge Computing
MN-CSE	Middle Node - Common Services Entity
MQTT	Message Queuing Telemetry Transport
NAT	Network Address Translation
PIP	Policy Information Point
PoA	Point-of-Access
PoC	Proof of Concept
QoS	Quality of Service
RDF	Resource Description Framework
RNI	Radio Network Information

SAREF	Smart Application REFerence ontology
SP	Service Provider
SUPI	Subscriber Permanent Identifier
TLS	Transport Layer Security
UDP	User Datagram Protocol
UE	User Equipment
UI	User Interface
WLAN	Wireless LAN
WSGI	Web Server Gateway Interface
XML	Extensible Markup Language

4 Overview

4.1 Introduction

The present document addresses the pivotal integration of oneM2M and Multi-access Edge Computing (MEC), a convergence deemed critical for enabling advanced Internet of Things (IoT) applications at the network edge. As the scale of IoT deployments expands and the demand for low-latency, real-time data processing intensifies, the seamless interworking of these two standardized frameworks becomes imperative. This clause establishes the foundational context for a detailed examination of how the oneM2M IoT service layer can effectively leverage the capabilities inherent in MEC for localized data processing and optimized operational efficiency.

The present document is structured to provide a comprehensive analysis of the integrated environment.

Clause 5 provides an overview of available the MEC Sandbox implementation, detailing its architectural characteristics and the specific functionalities offered for MEC application development and deployment at the edge. This includes an examination of how these platforms align with ETSI MEC specifications to enable localized computing and networking services.

Clause 6 subsequently delves into the landscape of available oneM2M implementations, specifically focusing on prominent open-source solutions such as ACME, Mobius and tinyIoT. This clause describes their adherence to oneM2M standards, their respective capabilities in managing IoT resources and services, and their suitability for various deployment models, including those relevant to edge environments.

Clause 7 is dedicated to the systematic analysis of key feature requirements for both MEC and oneM2M platforms (as elaborated in clauses 7.1 and 7.2), the proposed deployment options and associated use case selection (as defined in clause 7.3), and the establishment of assessment criteria pertinent to various implementations (as presented in clause 7.4). Following this comprehensive technical evaluation.

Clause 8 sets forth the principal recommendations and conclusions derived from the entirety of the analysis, with a particular emphasis on identifying suitable open-source implementations for Proof of Concept (PoC) development that effectively demonstrate the capabilities of edge-centric IoT use cases.

5 MEC Implementations

5.1 Introduction

Multi-access Edge Computing (MEC), standardized by the ETSI, is a transformative technology that brings cloud-computing capabilities and IT service environments to the edge of mobile networks. By moving computation and storage closer to end users, MEC enables ultra-low latency, high bandwidth, and real-time access to radio network information, unlocking new business opportunities and use cases across multiple sectors [i.1].

ETSI MEC is designed as an open standard, ensuring interoperability and supporting multiple implementations. It is access-agnostic, supporting 4G, 5G, Wi-Fi, and other network technologies, and aligns closely with the ETSI NFV (Network Functions Virtualisation) framework and 3GPP standards for seamless integration into modern telecom environments.

The MEC architecture consists of the following key elements [i.1]:

- MEC Platform: The software layer that provides the environment for MEC applications to run, offering APIs for service discovery, event notifications, and management.
- MEC Host: The physical or virtualized infrastructure at the edge that runs the MEC platform and applications.
- MEC Applications: Software components deployed at the edge to provide services such as video analytics, IoT data processing, or location-based services.
- RESTful APIs: Standardized interfaces that allow applications to interact with MEC services, such as location, radio network information, and application mobility.

The MEC architecture consists of the following key features [i.1]:

- Open APIs: Developer-friendly, standardized APIs to foster rapid application development and ensure consistency.
- Service Exposure: Real-time access to network and context information, including user location, network status, and device mobility.
- Multi-access Support: Operates across different access networks (cellular, Wi-Fi®, etc.).
- Ecosystem Enablement: Supports a wide range of verticals, including automotive, IoT, and smart cities, and encourages collaboration with open-source communities.

5.2 MEC Implementation #1 – ETSI MEC Sandbox

5.2.1 Overview

The ETSI MEC Sandbox is an interactive, cloud-based environment that enables developers, researchers, and solution architects to learn, experiment, and validate applications using ETSI MEC standardized service APIs. It is designed to bridge the gap between standards and real-world deployment by providing hands-on access to MEC APIs in a simulated but realistic network environment [i.2].

The ETSI MEC Sandbox key capabilities are [i.2]:

- API Experimentation: Developers can interact directly with live MEC Service APIs (RESTful, OpenAPI-compliant), including those for location (ETSI GS MEC 013 [i.7]), radio network information (ETSI GS MEC 012 [i.6]), WLAN information (ETSI GS MEC 028 [i.10]), application enablement (ETSI GS MEC 011 [i.5]), application mobility (ETSI GS MEC 021 [i.9]), V2X information (ETSI GS MEC 030 [i.13]), and federation (ETSI GS MEC 040 [i.14]).
- Scenario Simulation: Users can deploy pre-defined network scenarios combining 4G, 5G, and Wi-Fi technologies, configure terminal devices, and observe their behaviour on a geolocated map of Monaco.
- Dashboard Interface: The Sandbox provides a user-specific dashboard for managing network scenarios, terminal devices, MEC application instances, and service lifecycles. It includes tools for API testing (SwaggerUI), scenario configuration, and real-time visualization of assets and connectivity.
- External Integration: Developers can connect their own or third-party applications to the Sandbox, using provided API endpoints to test real-world interactions and notification subscriptions.
- Developer Resources: The MEC Sandbox is complemented by a comprehensive Wiki, offering documentation, ecosystem information, API conformance test suites, and proof-of-concept activities.
- Supported APIs and Services.

5.2.2 Services exposure

The ETSI MEC Sandbox exposes the following services:

- ETSI GS MEC 011 [i.5] (App Enablement): Application registration, service discovery, event notifications, traffic/DNS rules;
- ETSI GS MEC 011 [i.5] (MEC service management realized by CAPIF APIs): Common API Framework for 3GPP Northbound APIs [i.3];
- ETSI GS MEC 012 [i.6] (Radio Information): Radio network conditions, user plane measurements, UE context and events;
- ETSI GS MEC 013 [i.7] (Location Information): Geospatial and network location information, user tracking, zonal status;
- ETSI GS MEC 015 [i.8] (Traffic Management APIs): Bandwidth Management and Multi-access Traffic Steering;
- ETSI GS MEC 021 [i.9] (App Mobility): Support for application/user context relocation between MEC hosts;
- ETSI GS MEC 028 [i.10] (WLAN Information): WLAN access point and terminal information, connectivity events;
- ETSI GS MEC 029 [i.11] (Fixed network): Service access on Fixed Access Information for Fibre;
- ETSI GS MEC 030 [i.13] (V2X Information): Vehicle-to-everything information, predicted QoS, secure communication with 3GPP functions;
- ETSI GS MEC 033 [i.12] (IoT Information): Communication between the IoT devices and the IoT applications;
- ETSI GS MEC 040 [i.14] (Federation): MEC system/service/app instance discovery across federated edge/cloud systems.

5.2.3 How It Works

Upon signing in, users can:

- Deploy and configure network scenarios and terminal devices.
- Manage MEC application instance IDs.
- Enable or disable MEC services.
- Experiment with APIs via browser-based tools or external applications.
- Observe real-time changes and responses based on scenario state.
- The Sandbox emulates MEC platforms and orchestrates applications externally, requiring manual provisioning of application instance IDs for testing.

Here an example of the possible use cases that can be develop:

- Location-based services and analytics.
- Real-time video processing and content delivery.
- IoT data aggregation and processing at the edge.
- Connected vehicle and V2X applications.
- Smart city and industrial automation solutions.

5.2.4 Benefits for Developers and Industry

The key benefits the ETSI MEC Sandbox offers are:

- Rapid Prototyping: Accelerates the development and validation of edge applications by providing a ready-to-use, standards-compliant testbed.
- Interoperability Testing: Ensures applications are compatible with ETSI MEC APIs, reducing integration risks.
- Ecosystem Growth: Fosters innovation by lowering entry barriers for new developers and supporting collaboration across industry verticals.
- Realistic Scenarios: Enables testing in simulated environments that closely mimic real-world network conditions and user mobility.

5.2.5 License

- ETSI MEC Sandbox: Apache® license.
- MEC conformance test suites: BSD-3.

6 oneM2M Implementations

6.1 Introduction

The oneM2M standard provides a globally harmonized framework for Machine-to-Machine (M2M) and Internet of Things (IoT) service platforms, defining a Common Services Entity (CSE) architecture that supports interoperability across heterogeneous devices and networks. A variety of oneM2M-compliant implementations have emerged to support research, prototyping, deployment, and validation of real-world IoT solutions.

Open-source implementations play a crucial role in accelerating the adoption of the oneM2M standard by providing accessible platforms for experimentation, integration with vertical applications, and demonstration of conformance. These implementations vary in terms of target environment, supported features, technology stack, and extensibility.

This clause introduces three representative open-source oneM2M CSE implementations that are actively maintained and widely adopted in the community:

- **ACME:** A Python®-based CSE implementation that prioritizes developer experience, extensibility, and testability.
- **tinyIoT:** A lightweight C-language implementation optimized for edge computing scenarios and resource-constrained devices.
- **Mobius™:** A scalable Java® Node.js-based platform developed under the OCEAN project, suited for smart city IoT deployments.

Each implementation is analysed in terms of specification conformance, supported resource types, protocol bindings, runtime features, installation procedures, and relevance to MEC integration. The goal is to assist developers and system integrators in selecting the most appropriate implementation for their PoCs, research projects, or commercial deployments.

6.2 oneM2M Implementation #1 – ACME oneM2M CSE

6.2.1 Overview

The ACME oneM2M CSE (ACME CSE for short) is an implementation of a oneM2M *Common Service Entity* that supports a rich subset of the oneM2M IoT standard. Its purpose is to provide an easy to install and run oneM2M CSE. The ACME CSE is written in Python, and can be installed and run with a few commands almost everywhere where the Python runtime environment is available. By default, the implementation uses a simple file-based document database for data storage that is suitable for small installations. For more sophisticated deployments, the ACME CSE can connect to and use a PostgreSQL database installation.

In addition to a Web-based UI, the ACME CSE also offers a rich text-based user interface that runs directly inside a terminal console. It provides a convenient way to inspect and work with resources, requests, and status information. This UI is especially useful when running the CSE on a remote server. The ACME CSE is designed to be extensible. It is possible to add new resource types, and new service and runtime functions.

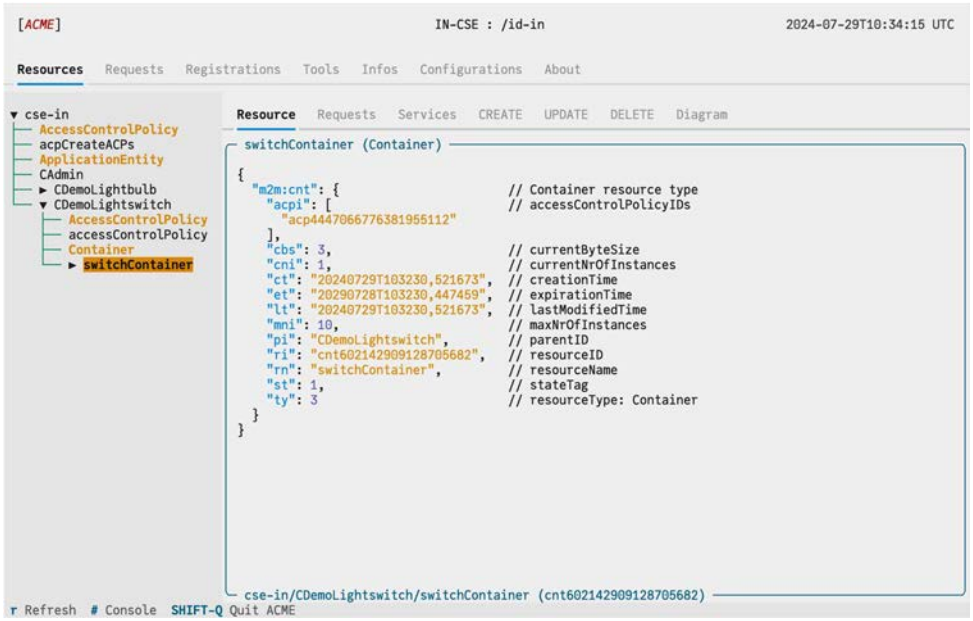

Figure 6.2.1-1: ACME CSE: Text UI

A more basic console UI is also available. It is the default UI when running the ACME CSE in a terminal console, and which is better suited to show the extensive log and debug output. The ACME CSE is under active development to include further functionalities as well as to trial new and experimental functions that are currently under discussion for the oneM2M specification.

It is available on GitHub under the BSD 3-Clause License:

- Homepage and Documentation: https://acmecse.net
- GitHub Repository: https://github.com/ankraft/ACME-oneM2M-CSE

6.2.2 Supported oneM2M Feature

6.2.2.1 oneM2M Specification Conformance

The ACME CSE successfully passes all the relevant oneM2M test cases for the supported resource types, attributes and behaviours.

6.2.2.2 Release Versions

The ACME CSE supports oneM2M release 1 - 4 and the upcoming release 5 for the supported resource types and functionalities listed below.

6.2.2.3 CSE Types

The ACME CSE supports the following CSE types:

- IN-CSE - Infrastructure Node CSE
- MN-CSE - Middle Node CSE
- ASN-CSE - Application Service Node

6.2.2.4 Resource Types

The ACME CSE supports the oneM2M resource types listed in table 6.2.2.4-1. Some resource types are not yet fully implemented, and some features are experimental.

Table 6.2.2.4-1: ACME CSE: Supported resource types

Resource Type	Remark
Entities	
CSEBase (CB)	The CSEBase resource type is fully supported.
Application Entity (AE)	The Application Entity resource type is supported, except for some of the "S" registration related functionalities.
RemoteCSE (CSR)	The ACME CSE supports CSE registrations via the Mcc reference point. In addition announced resources, synchronization, and transit requests that target resources on remote CSE's are supported, as well.
Security	
Access Control Policy (ACP)	In addition to the basic ACP functionality, the ACME CSE supports the following advanced ACP features: - Attribute-based access control - AccessControlWindow
Data Management	
Container (CNT)	The Container resource type is fully supported.
ContentInstance (CIN)	The ContentInstance resource type is fully supported. Values with the following data types are supported for the content attribute: - string - integer - float - boolean - list - dictionary / JSON object
FlexContainer (FCNT)	The ACME CSE fully supports the FlexContainer resource type. It is possible to use the predefined FlexContainer Specializations of ETSI TS 118 123 [i.17], *AllJoin*, oneM2M's *GenericInterworking*, or to define custom FlexContainerSpecializations.
FlexContainerInstance (FCI)	This is an experimental implementation of the draft FlexContainerInstance specification.
TimeSeries (TS)	The ACME CSE supports missing data detection and the respective notifications.
TimeSeriesInstance (TSI)	The TimeSeriesInstance resource type is fully supported, except for the dataGenerationTime attribute, which is only supported with absolute timestamps. Values with the following data types are supported for the content attribute: - string - integer - float - boolean - list - dictionary / JSON object

Subscription and Notification	
CrossResourceSubscription (CRS)	Besides the standard CrossResourceSubscription functionality the ACME CSE implements an experimental feature to support an *eventEvaluationMode* to react on missing events.
Subscription (SUB)	The ACME CSE supports notifications via direct url or an AE's Point-of-Access (POA). Further, BatchNotifications, attributes, notification statistics, and operation monitoring are supported.
Device Management	
Node (NOD)	The Node resource type is fully supported.
Management Objects	The ACME CSE supports the following management objects: - AreaNwkDeviceInfo (ANDI) - AreaNwkInfo (ANI) - Battery (BAT) - Credentials (CRDS) - DataCollect (DATC) - DeviceCapability (DVC) - DeviceInfo (DVI) - EventLog (EVL) - Firmware (FWR) - Memory (MEM) - MobileNetwork (MNWK) - MyCertFileCred (NYCFC) - Reboot (REB) - SIM (SIM) - Software (SWR) - WifiClient (WIFIC)
Communication	
PollingChannel (PCH)	Request and notification long-polling via the *pcu*(*pollingChannelURI*) virtual child resource are supported. *requestAggregation* functionality to retrieve multiple requests in one polling request is supported as well.
Request (REQ)	The ACME CSE supports blocking, and synchronous and asynchronous non-blocking requests, which are managed through <request> resources.
Group Management	
Group (GRP)	The ACME CSE supports requests via the *fopt* (fanOutPoint) virtual resource. Remote resources may be members of a group.
Automation	
Action (ACTR)	The *input* attribute of the <action> resource type is not supported yet.
Dependency (DEPR)	The Dependency resource type is fully supported.
Semantics	
SemanticDescriptor (SMD)	The ACME CSE supports semantic queries and discovery for resources with semantic descriptors. currently, the following presentation formats are supported: - RDF/XML - JSON-LD - Turtle
Location Management	
LocationPolicy (LCP)	Only device-based location policy is supported. The LCP's <container> resource stores geo-coordinates and geo-fencing results.
Time Management	
Schedule (SCH)	The ACME CSE supports scheduling for various of its functions, such as CSE communication windows, and resource types, such as the <node>, <subscription> and <crossResourceSubscription> resource types.
TimeSyncBeacon (TSB)	The ACME CSE supports the TimeSyncBeacon resource type. The functionality is currently only experimental and might change according to specification changes.

Figure 6.2.2.4-1 shows the class hierarchy of the supported resource types.

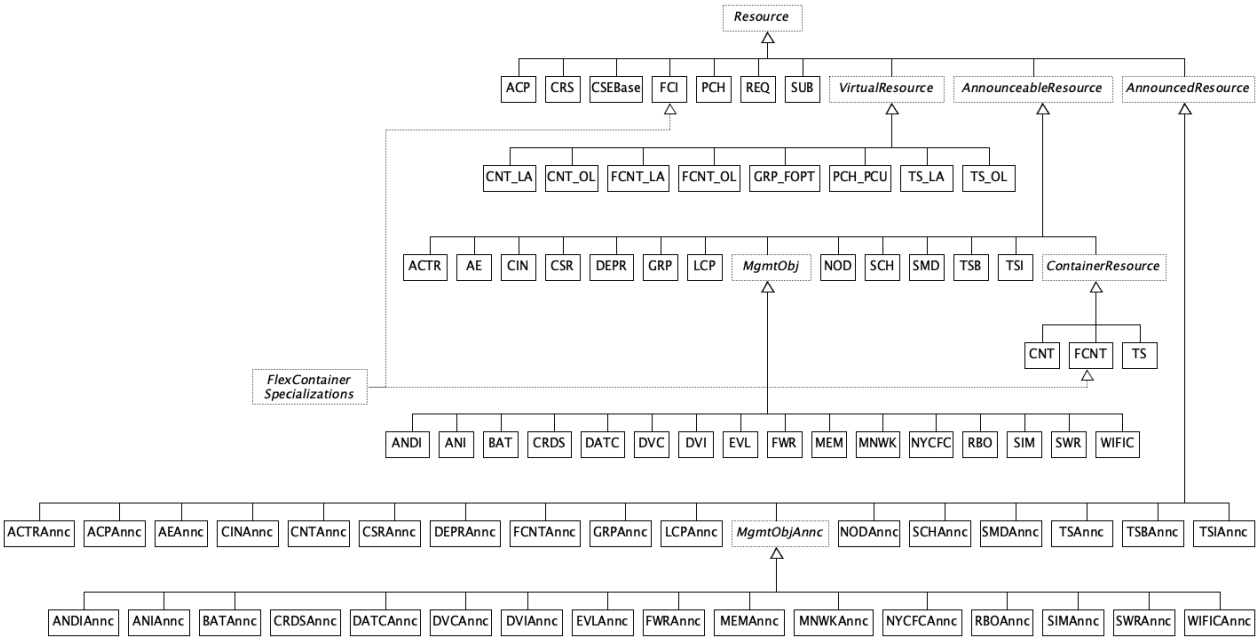

Figure 6.2.2.4-1: ACME: Hierarchy of the supported oneM2M resource types

6.2.2.5 Service Functionalities

Table 6.2.2.5-1 list the supported oneM2M service functionalities.

Table 6.2.2.5-1: ACME CSE: Supported oneM2M service functionalities

Service Functionality	Remark
AE Registration	The ACME CSE supports AE registration and de-registrations via the Mca reference point.
Blocking and Non-Blocking Requests	Blocking requests are the common way for an AE to interact with the CSE. They are also used for CSE-to-CSE communication via the Mcc reference point. To avoid blocking the AE, the ACME CSE supports non-blocking requests in synchronous and asynchronous mode as well.
Delayed Request Execution	The ACME CSE supports delayed request execution via the *Operation Execution Timestamp* request parameter.
Discovery	The ACME CSE supports normal retrieval and discovery of resources via the *filterCriteria* and *discoveryResultType* parameters.
Geo-Query	The ACME CSE supports location and geometry queries and location-based discovery.
Location Management	Only *device based*, and *no network-based* location policies are supported.
Long Polling	*Long Polling* is supported for request unreachable AEs and CSEs through <*pollingChannel*> resources. This mechanism is used for sending notifications and other requests to AEs and CSEs that are not directly reachable via the Mca or Mcc reference points, e.g. because of firewalls or NATs.
No-Response Requests	*No-response* requests are supported for requests that do not require a response. This is especially useful for constrained devices that do not need to wait for a response, or cannot handle possible error responses. This feature is still experimental and might change according to specification changes.
Notifications	The ACME CSE supports notifications via direct url, or an AE's or CSE's Point-of-Access (PoA). Further, *batchNotifications*, attributes, notification statistics, and operation monitoring are supported.

Service Functionality	Remark
Notification Event Types	The following notification event types for Subscriptions are supported: - resourceUpdate - createDirectChild - deleteDirectChild - blockingUpdate - retrieveCNTNoChild - missingData - triggerReceivedForAE
Partial Retrieve	Partial retrieve of individual resource attributes is supported.
Remote CSE Registration	The ACME CSE supports CSE registrations via the Mcc reference point. In addition, announced resources, synchronization, and transit requests that target resources on remote CSE's are supported.
Request Expiration	The *Request Expiration Timestamp* request parameter is supported.
Request Forwarding	Forwarding requests from one CSE to another is supported.
Request and Resource Validations	All requests received via the Mca and Mcc reference points are validated.
Resource Addressing	CSE-Relative, SP-Relative and Absolute as well as hybrid addressing are supported.
Resource Announcements	Announcements of resources are supported via the Mcc reference point. Resources are announced under a <CSEBaseAnnc> resource on the target CSE (R4 feature). Bi-directional update and attribute synchronization is supported as well.
Resource Expiration	Resources are automatically deleted after the expiration time has passed.
Result Content Types	The following result contents are implemented for standard oneM2M requests and discovery: - nothing - attributes - hierarchical address - hierarchical address + attributes - attributes + child-resources - attributes + child-resource-references - child-resource-references - original-resource - child-resources - modified attributes - semantic content - discovery result references
Result Expiration	The *Result Expiration Timestamp* request parameter is supported.
Semantics	Basic support for semantic descriptors, semantic queries and discovery is available.
Subscriptions	Subscriptions are supported, including batch notification, and resource type and attribute filtering.
Time Synchronization	The ACME CSE supports time synchronization via the *<timeSyncBeacon>* resource type.
TimeSeries Data Handling	<TimeSeries> data handling is supported, including missing data detection, monitoring and notifications.

6.2.2.6 Bindings

The following Protocol Bindings are supported. It is possible to enable only one binding, or to use any combination of them for a CSE instance.

Table 6.2.2.6-1: ACME CSE: Supported oneM2M protocol bindings

Protocol Binding	Remark
CoAP	The CoAP protocol is supported. CoAP over DTLS is not yet implemented. The current implementation only supports the CoAP binding specification for oneM2M Release 1-4. The upcoming Release 5 specification is under development.
HTTP	The http protocol is supported, including TLS (https), CORS and WSGI support. The *basic* and *bearer* authentication methods are supported.
MQTT	The mqtt protocol is supported, including TLS (mqtts) support. This protocol binding requires a separate mqtt broker to be installed and running. Basic username/password authentication with an mqtt broker is supported as well.
WebSocket	The WebSocket protocol is supported, including TLS (wss) support.

6.2.2.7 Types

The following serialization types for requests and responses are supported.

Table 6.2.2.7-1: ACME CSE: Supported oneM2M serializations

Serialization Type	Remark
CBOR	CBOR serialization is supported.
JSON	In addition to the common JSON syntax, C-style and Python-style comments are supported as well: Block comment: /* ... */ End-of-line comment: // ... and # ...

6.2.3 CSE Runtime Features

6.2.3.0 Overview

In addition to the oneM2M standard functionalities, the ACME CSE implements additional features to help with deployments and to understand and learn about oneM2M.

6.2.3.1 Database Bindings

The following database bindings are supported.

Table 6.2.3.1-1: ACME CSE: Supported database bindings

Serialization Type	Remark
PostgreSQL	PostgreSQL is a powerful, open-source object-relational database system. The ACME CSE can be configured to use a local or remote PostgreSQL database for data storage.
TinyDB "file-based"	TinyDB is a fast and lightweight document-oriented database that is ideal for small installations.
TinyDB "in-memory"	TinyDB can also be used as an in-memory database. This is useful for testing and small installations. This database binding offers the fastest processing speed and response times. However, it offers no persistence of data between restarts.

6.2.3.2 Additional Runtime Features

The ACME CSE provides the following additional features.

Table 6.2.3.2-1: ACME CSE: Additional Runtime Features

Serialization Type	Remark
Guided Setup	An interactive setup process that guides the user through the initial configuration of the CSE.
HTTP Authorization	Basic support for *basic* and *bearer* (token) authorization.
HTTP CORS	Support for *Cross-Origin Resource Sharing* to support http(s) redirects
HTTP WSGI	Support for the *Web Server Gateway Interface* to improve integration with a reverse proxy and API gateway, i.e. Nginx.
Recording Requests	Requests over Mca and Mcc to and from a CSE can be recorded. This may be used to inspect communication sequences between oneM2M entities and to debug requests.
Script Interpreter	The CSE includes a Lisp-based script interpreter to extent CSE functionalities, implement simple AEs, prototypes, tests, and more.
Testing: Upper Tester	Basic support for the Upper Tester protocol defined in ETSI TS 118 119 [i.18], and additional command execution support.
Text Console	Control and manage the CSE, inspect resources, and run scripts in a text console. The log output is also displayed in the text console.
Text UI	A text-based user interface to inspect resources and requests, configurations, stats, manage resources, display diagrams, and more.
Web UI	A Web UI that displays the oneM2M resource tree and offers a basic REST UI.

6.2.4 System Requirements

The ACME CSE requires Python 3.10 or newer. The implementation has been tested with the new Python version 3.13, but some of the third-party components do yet not fully comply with the new experimental *free-threaded* mode of the Python runtime. It is therefore recommended to not enable this runtime mode.

Supported runtime environments include:

- Generic Linux® Including Raspberry Pi OS™ (32bit or 64bit) on Raspberry Pi™, for example on RaspberryPi™ Zero W, RaspberryPi™ Zero 2 W, RaspberryPi™ 3 - 5
- MacOS®
- MS Windows®
- Jupyter Notebooks™
- The ACME CSE can be run headless inside a Jupyter Notebook™, and receive and handle local requests
- Docker®
- The ACME CSE can be run in a Docker® container

6.2.5 Installation, Setup and Running

6.2.5.0 Overview

The ACME CSE can be either installed via the Python *pip package manager* (*pip* is Package Installer for Python (https://pip.pypa.io)) or manually. In both case it is highly recommended to use a virtual environment, such as virtualenvs, pyenv or uv to keep Python package installations separate.

6.2.5.1 PIP Installation

Run *pip* to install the latest ACME CSE release from the Python Package Index (PyPI):

```
$ python -m pip install acmecse
```

This will install the latest version of the ACME CSE and all required dependencies. One can also upgrade to the latest version by running:

```
$ python -m pip install --upgrade acmecse
```

6.2.5.2 Manual Installation

This is an alternative to the installation method described in clause 6.2.5.1.

One can install the ACME CSE by cloning the repository, or by downloading the latest release package, unpacking it, and copying the whole distribution to a new directory.

```
$ git clone https://github.com/ankraft/ACME-oneM2M-CSE.git
$ cd ACME-oneM2M-CSE
```

The next step installs the additional packages that are required. It is recommended to install the required packages by running the following command:

```
$ python3 -m pip install -r requirements.txt
```

6.2.5.3 Additional Components

Depending on the deployment requirements it may be necessary to install separate components.

- **MQTT Broker**
  A separate MQTT broker is required to use the oneM2M MQTT protocol binding to connect via the MQTT protocol. The ACME CSE has been tested with Eclipse Mosquitto™ (see https://mosquitto.org).

- **PostgreSQL Database System**
  To store oneM2M resources and other runtime data in another database as the built-in file-based document database a separate database system is required. The ACME CSE can be configured to use the PostgreSQL database system as a document database.

6.2.5.4 Configuration and Onboarding

The ACME CSE is highly configurable and can be adapted to different environments and requirements. Configuration of CSE parameters is primarily done through a configuration file. This file contains all configurable and customizable settings for the CSE. Configurations are mostly optional, and settings in this configuration file overwrite the ACME CSE's default values.

Some configuration values can also be overwritten via command line arguments or by environment variables. This is especially useful when running the ACME CSE in a Docker environment.

When the ACME CSE is run for the first time and no configuration file can be found then an interactive onboarding process is started where the user is guided to enter the most important configuration values. At the end of this process the configuration file is created, and the ACME CSE starts.

More detailed information about the configuration settings can be found at https://acmecse.net/setup/Configuration-introduction.

6.2.5.5 Running the ACME CSE

One can start the ACME CSE by simply running it from the command line.

Running after PIP installation from any directory:

```
$ acmecse
```

Running after manual installation from the source directory:

```
$ python3 -m acme
```

The ACME CSE is usually started with the text console interface that provides extensive information and logging information. The text UI is then started by pressing the "#" key text UI. To automatically open the text UI at startup, one can provide the "--texui" command line argument.

The documentation at https://acmecse.net/setup/Running provides further information about available command line arguments and other running help.

6.2.6 Test Suite

The ACME CSE project provides an extensive set of test cases that cover the supported oneM2M and runtime functionalities. This test suite is available in the sub-directory "tests" after performing the manual installation described above. Though designed to continuously test features and performance of the ACME CSE the text suite can be used with other oneM2M CSE implementations as well.

The detailed configuration and usage of the ACME CSE test suite is described in detail at:

https://acmecse.net/development/UnitTests/.

6.3 oneM2M Implementation #2 – tinyIoT

6.3.1 Overview

tinyIoT is a lightweight, open-source oneM2M Common Services Entity (CSE) implementation designed for constrained IoT edge environments. It focuses on essential core features of the oneM2M standard and aims to support low-resource platforms while enabling easy deployment, interoperability, and extensibility.

Written in C and optimized for Linux-based systems, tinyIoT is suitable for edge computing scenarios, smart device gateways, and embedded Linux platforms such as Raspberry Pi and industrial gateways. It provides native oneM2M resource handling, supports RESTful communication, and is designed for low-latency, high-reliability deployments.

tinyIoT emphasizes modularity and performance. It includes lightweight internal modules for CoAP, MQTT, HTTP, SQLite-based storage, and a plugin interface for AI model integration or data stream processing. Its minimal footprint makes it particularly suitable for experimentation with distributed edge intelligence or interworking with MEC platforms.

6.3.2 Supported oneM2M Features

6.3.2.1 oneM2M Specification Conformance

tinyIoT is compliant with core features of oneM2M Release 2 and partially supports Release 3, focusing on:

- AE registration and management
- Resource Creation, Retrieval, Update, Deletion (CRUD)
- Container/ContentInstance resources
- Subscription and notification handling
- CSE registration and interworking

6.3.2.2 CSE Types

- IN-CSE
- MN-CSE
- ASN-CSE

6.3.2.3 Resource Types

The following oneM2M resource types are supported in the current version of tinyIoT.

Table 6.3.2.3-1: Resource types supported by tinyIoT

Resource Type	Supported	Description
`CSEBase`	Support	Root node of the CSE hierarchy
`AE (Application Entity)`	Support	Application Entity
`Container (CNT)`	Support	Logical container for content
`ContentInstance (CIN)`	Support	Actual data item stored under containers
`Subscription (SUB)`	Support	Supports notification via HTTP, CoAP, MQTT and WebSocket
`RemoteCSE (CSR)`	Support	Remote CSE registration and forwarding
`Group (GRP)`	Support	Management of multiple child resources
`AccessControlPolicy (ACP)`	Support	Basic access control enforcement
`TimeSeries (TS)`	Support	Time-series data stream resource
`TimeSeriesInstance (TSI)`	Support	Data entries under a TimeSeries
`flexContainer (FCNT)`	Plan to support	Support custom data types via defined schemas

Resources such as `Node`, `SemanticDescriptor`, and `LocationPolicy` are currently not supported, but are planned for future releases.

6.3.3 Functional Capabilities

tinyIoT supports the following core oneM2M service functionalities:

- CRUD operations on supported resource types
- AE registration and deregistration
- Resource discovery using filterCriteria
- Cross-CSE registration and forwarding via CSR
- Resource announcement between CSEs
- Event-based notification via HTTP, MQTT, and WebSocket
- Access control using ACP policies
- Group-based fan-out operations via GRP
- Basic time-series data handling with support for monitoring and structured insertion

Not yet supported:

- Cross-resource subscriptions
- Long polling
- Semantic discovery
- Operation monitoring

6.3.4 Protocol Bindings

tinyIoT provides support for the following oneM2M protocol bindings.

Table 6.3.4-1: oneM2M protocols binding

Protocol	Status	Notes
HTTP	Support	Support RESTFul API
CoAP	Support	Targeting constraint devices
MQTT	Support	Requires external broker
WebSocket	Support	Enables low-latency, full-duplex communication

These bindings allow flexibility in connecting constrained devices and edge platforms in various network environments.

6.3.5 Serialization Formats

Table 6.3.5-1: oneM2M serialization formats

Format	Status	Notes
JSON	Support	Primary format, used for all communication
XML	Support	Alternative data format

6.3.6 Runtime Architecture

- Language: C
- Database: SQLite (lightweight, persistent)
- Configuration: `.ini` file or environment variable override
- Logging: Built-in structured logger
- Plugin Support: Modular architecture supports dynamically loaded libraries for edge AI, data processing, and interworking modules
- Deployment: Native binary or Docker container on Linux environments

6.3.7 MEC Integration

tinyIoT is well-suited for Multi-access Edge Computing (MEC) deployment:

- Can run as a MEC Application with minimal footprint
- Supports hosting of oneM2M MN-CSEs at the edge
- Supports service migration via announcement + resource handover (planned)
- Lightweight enough to operate within virtualized containers on MEC platforms
- Designed to be enhanced with MEC-related APIs (e.g. Location API, RNI API) in future updates

6.3.8 Installation, Configuration, and Execution

6.3.8.1 Installation

tinyIoT can be installed from source on any Linux-based system. It has minimal dependencies and supports both native and cross-compilation.

Steps for building and installing:

```
# Clone the repository
$ git clone https://github.com/seslabSJU/tinyIoT.git
$ cd tinyIoT

# Build the project
$ make
```

tinyIoT can also be compiled for embedded systems (e.g. ARM-based platforms) by configuring the `Makefile` with the appropriate cross-compiler settings.

6.3.8.2 Configuration

tinyIoT is configured via a simple .ini-style configuration file (e.g. `config.ini`), which defines:

- Network port and IP bindings
- Protocol bindings (HTTP, MQTT, WebSocket)
- Logging level and output
- SQLite database location
- Resource tree root parameters
- Feature toggles (e.g. WebSocket, time-series)

Example `config.ini` snippet:

```
[server]
http_port = 8080
enable_websocket = true

[database]
db_path = ./tinyiot.db

[logging]
level = info
```

Environment variables can override configuration file values for container-based or automated deployments.

6.3.8.3 Running the CSE

Once compiled and configured, tinyIoT can be launched using the following command:

```
$ ./cse config.ini
```

Upon launch, tinyIoT initializes the CSEBase, loads persisted resources (if any), and opens all configured protocol endpoints (e.g. HTTP server, WebSocket listener). Logging output is shown in the console for diagnostics.

6.3.8.4 Docker Support

A basic Dockerfile is provided in the repository for containerized deployments:

```
# Build the container image
$ docker build -t tinyiot .

# Run the container
$ docker run -p 8080:8080 tinyiot
```

This enables integration with cloud-native platforms or edge orchestration systems (e.g. ETSI MEC platforms).

6.3.9 Version and Repository Information

- Current Version: v0.9.3

NOTE: This is based on the latest tagged or documented commit as of May 2025; subject to update in the repository).

- License: MIT License
- Source Repository:
  https://github.com/seslabSJU/tinyIoT
- Maintainer:
  Smart Embedded Systems Lab (SESLAB), Sejong University, Korea
  https://seslab.sejong.ac.kr
- Documentation:
  Basic README and configuration instructions are provided in the repository. Detailed developer documentation, build guides, and integration examples are currently under development.
- Release Roadmap Highlights:
  - v1.0.0 (planned): Full support for oneM2M Release 3 core features
  - Integration with ETSI MEC Location/RNI APIs (planned in the ESTIMED project)
  - Extended plugin architecture for AI-based edge inference
  - CSE-to-CSE resource synchronization enhancements

6.4 oneM2M Implementation #3 – Mobius

6.4.1 Overview

Mobius is an open-source oneM2M Common Service Entity (CSE) developed by the Korea Electronics Technology Institute (KETI) as part of the OCEAN project. Written in JavaScript and running on Node.js, Mobius is designed to provide a scalable and flexible IoT platform that complies with oneM2M standards. It supports various communication protocols and is suitable for deployment in diverse environments, from cloud servers to edge devices.

6.4.2 Supported Resource Type

Mobius supports a wide range of oneM2M resource types, enabling comprehensive IoT service development.

Table 6.4.2-1: oneM2M resource types for Mobius

Resource Type	Support Status	Description
CSEBase	Support	Root resource of the CSE
AE (Application Entity)	Support	Represents applications
Container (CNT)	Support	Logical containers for data
ContentInstance (CIN)	Support	Actual data instances
Subscription (SUB)	Support	Enables event notifications
RemoteCSE (CSR)	Support	Represents remote CSEs
Group (GRP)	Support	Groups multiple resources
AccessControlPolicy (ACP)	Support	Manages access permissions
TimeSeries (TS)	Support	Handles time-series data
FlexContainer (FCNT)	Plan to support	Customizable containers
Node (NOD)	Support	Represents physical devices
SemanticDescriptor (SMD)	Support	Add semantic information

6.4.3 Functional Capabilities

Mobius offers a comprehensive set of functionalities aligned with oneM2M specifications:

- **CRUD Operations**: Create, retrieve, update, and delete resources.
- **Resource Discovery**: Locate resources based on specific criteria.
- **Subscription and Notification**: Subscribe to resource changes and receive notifications.
- **Resource Announcement**: Share resources across different CSEs.
- **Access Control**: Define and enforce access permissions using ACPs.
- **Time-Series Data Management**: Handle sequential data efficiently.
- **Semantic Support**: Incorporate semantic descriptors for enhanced interoperability.
- **Group Management**: Manage multiple resources collectively.
- **Location Management**: Associate resources with location information.

6.4.4 Protocol Bindings

Mobius provides support for the following oneM2M protocol bindings.

Table 6.4.4-1: Protocol bindings supported by Mobius

Protocol	Status	Notes
HTTP	Support	Support RESTFul API
CoAP	Support	Targeting constraint devices
MQTT	Support	Requires external broker
WebSocket	Support	Enables low-latency, full-duplex communication

These bindings allow flexibility in connecting constrained devices and edge platforms in various network environments.

6.4.5 Serialization Formats

Table 6.4.5-1: Serialization formats supported by Mobius

Format	Status	Notes
JSON	Support	Default data format
XML	Support	Alternative data format

6.4.6 Installation and Configuration

Prerequisites:

- **Node.js:** JavaScript runtime environment.
- **MySQL:** Relational database for storing resource data.
- **Mosquitto:** MQTT broker for message handling.

Installation Steps:

1) Clone the Repository:

```
git clone https://github.com/IoTKETI/Mobius.git
cd Mobius
```

2) Install Dependencies:

```
npm install
```

3) Configure MySQL:

- Install MySQL server.
- Import the database schema:

```
mysql -u root -p < mobius/mobiusdb.sql
```

4) Configure Mobius:

- Edit the `conf.json` file to set parameters such as:

```
{
  "csebaseport": "7579",
  "dbpass": "your_mysql_password"
}
```

5) Run Mobius:

```
node mobius.js
```

6.4.7 Repository and Version Information

- **Repository**: https://github.com/IoTKETI/Mobius
- **License**: BSD-3-Clause
- **Latest Release**: Refer to the GitHub repository for the most recent version and release notes.

7 Required Features for MEC and oneM2M

7.1 Features for MEC

7.1.0 Overview

To enable efficient integration of MEC and oneM2M platforms, the MEC platform should provide a set of key features and functionalities. These features will allow oneM2M platforms to interact seamlessly with MEC-hosted applications and services, ensuring that edge applications can leverage oneM2M data and resources while maintaining low-latency processing, and efficient communication. The features encompass device and platform management, real-time data access, service discovery, and protocol support, facilitating interoperability across diverse IoT environments.

7.1.1 IoT Device and Platform Management

The IoT API, as defined in ETSI GS MEC 033 [i.12], offers a RESTful interface to facilitate the integration and management of IoT devices and platforms within the Multi-access Edge Computing (MEC) environment. It provides essential functionalities for IoT system administrators and MEC applications, including device provisioning (registration, deregistration, and updates), IoT platform discovery, and transport configuration. The API supports operations to manage device identifiers (e.g. IMEI, SUPI, deviceId) and associate devices with traffic rules or user transports (e.g. MQTT, HTTP), enabling seamless communication with IoT platforms like oneM2M. Additionally, it allows querying and updating of IoT platform information, ensuring discoverability of native services via APIs.

7.1.2 Real-Time Sensor Data Access

The Sensor-sharing API, as defined in ETSI GS MEC 046 [i.15], provides a RESTful interface to enable MEC applications to interact with sensors in a multi-access edge computing environment. It offers functionalities for sensor discovery, allowing applications to query available sensors based on type, properties, or location; sensor status provision, providing real-time status updates (e.g. ON, OFF, ERROR, etc.); and sensor data provision, enabling retrieval of the latest sensor measurements. Additionally, the API supports sensor management, facilitating remote configuration of sensor parameters like sampling rate. Leveraging a publish/subscribe model alongside lookup mechanisms, the API ensures efficient data exchange, aligns with oneM2M and SAREF ontologies for interoperability.

7.1.3 Service Discovery and Consumption

Service discovery and consumption in MEC, as outlined in ETSI GS MEC 011 [i.5], provide MEC applications with the ability to find and utilize services offered by the MEC platform, enhancing their functionality at the network edge. These features allow applications to access MEC services, such as bandwidth management, and location-based capabilities, which enhance functionalities of applications running at the Edge. By enabling local interactions and preprocessing of application's data (i.e. IoT data) before cloud transmission, these services improve efficiency and performance for edge-based operations (i.e. IoT operations), supporting a wide range of applications with enhanced, real-time capabilities.

7.1.4 Network Resource Optimization

The Bandwidth Management (BWM) API, as defined in ETSI GS MEC 015 [i.8], enables MEC applications to efficiently manage bandwidth resources for optimized performance. It allows applications to register, update, or unregister bandwidth allocations, specifying size and priority for either entire application instances or specific sessions, using RESTful HTTP methods (GET, PUT, PATCH, POST, DELETE). The API supports a subscription-notification model, delivering real-time updates on bandwidth changes and data volume dispersion per User Equipment (UE) through JSON data structures like BwInfo and BwChgEventNotification. By facilitating dynamic bandwidth adjustments, the BWM API enhances Quality of Experience (QoE) for edge applications, such as IoT applications, Content Delivery Networks (CDNs) and gaming, ensuring efficient resource utilization and improved user performance.

7.1.5 Application Lifecycle Management

Application lifecycle management [i.4] in Multi-access Edge Computing (MEC) enables the creation, operation, and removal of applications at the network edge, ensuring fast and reliable services for users. It provides a structured way to launch applications, manage their active or inactive states, and efficiently end them when no longer needed, optimizing resources and supporting seamless performance. This process supports a wide range of applications, such as IoT, gaming and Sandbox environments for testing, by ensuring they can be quickly deployed and managed across different MEC systems, enhancing user experience with low-latency and high-performance services.

7.1.6 Required Features Summary

For seamless integration between MEC and oneM2M, the MEC platform should comprehensively support several key features.

Below is a summary of the key MEC features required for oneM2M interworking.

Table 7.1.6-1: Summary of key required features of MEC platform to support oneM2M

Key Features	Description
IoT Device and Platform Management	Enables local provisioning, configuration, and management of IoT devices and oneM2M platforms.
Real-Time Sensor Data Access	Ensures fast and reliable access to sensor data and events.
Service Discovery and Consumption	Allows MEC applications to discover and consume MEC and oneM2M services.
Network Resource Optimization	Supports dynamic bandwidth allocation and multiple transport protocols.
Application Lifecycle Management	Enables instantiation, scaling, and termination of applications.

Below is a summary of the key MEC specifications supporting the selected features.

Table 7.1.6-2: Key MEC specifications

Specification	Description
ETSI GS MEC 033 [i.12] (IoT API)	Provides APIs for provisioning and managing IoT devices and platforms
ETSI GS MEC 046 [i.15] (Sensor-sharing API)	Facilitates access to and sharing of sensor data
ETSI GS MEC 011 [i.5] (Edge Platform Application Enablement)	Defines service discovery and consumption mechanisms
ETSI GS MEC 015 [i.8] (Bandwidth Management API)	Enables dynamic allocation of network resources
ETSI GS MEC 010-2 [i.4] (Application Lifecycle Management)	Specifies lifecycle management processes for applications
ETSI GS MEC 003 [i.19] (Framework and Reference Architecture)	Provides the architectural framework for hosting applications and MEC services

7.2 Features for oneM2M

7.2.0 Data and Resource Management

To enable efficient integration of oneM2M and MEC platforms, the oneM2M platform should provide a set of key features and functionalities. These features will allow MEC platforms to host oneM2M edge platforms as its application and ensure that edge applications can take advantage of the data provided by oneM2M platforms while maintaining security and efficient communication.

7.2.1 Data and Resource Management

- Data management using CRUD (Creation, Retrieve, Update and Deletion): The oneM2M platform should allow for the creation, retrieve, update and deletion of various IoT resources (e.g. sensors, devices, measurements) through the use of container and contentInstance resources. These resources are essential for storing and managing data coming from different IoT devices in the network.

- Data Format Standardization: One of the key features of oneM2M is the ability to standardize data formats for interoperability between devices. To achieve this, the oneM2M platform should support XML or JSON data formats, which are widely adopted for data exchange in IoT systems. This ensures that devices, applications, and platforms can communicate effectively and consistently, regardless of underlying hardware or software variations.

- Time-Series Data Handling: The oneM2M platform should be capable of handling time-series data, which involves managing historical data retention along with the most recent measurements. This capability allows for the storage of time-stamped data over extended periods, making it possible to retrieve and analyse trends, perform forecasting, and conduct retrospective analyses. Specifically, the timeSeries attribute allows for efficient storage and querying of time-series data, while the latest attribute ensures that the most recent data points are always available for real-time use cases.

7.2.2 Real-Time Data Exchange

- Subscription and Notification: The oneM2M platform should provide a robust subscription/notification mechanism to enable applications and Common Service Entities (CSEs) to react to changes and events within the oneM2M system in real-time. This mechanism is fundamental for creating dynamic and responsive IoT solutions. The core of the mechanism is the <subscription> resource. An application (Application Entity, AE) or another CSE interested in monitoring changes to a particular resource creates a <subscription> resource as a child of the resource it wants to monitor (the "subscribed-to" resource).

- Bi-directional Announcement: In oneM2M, bi-directional announcement refers to the mechanism where a resource or information is made available across multiple Common Service Entities (CSEs) and can be updated or synchronized in both directions, from the original resource's CSE to the announced CSE and vice versa. This allows for a more dynamic and interconnected system where changes in one location can be reflected in another, and vice versa, facilitating a more flexible and responsive network. It is particularly useful in edge computing scenarios. Resources (or their representations) can be offloaded to edge CSEs closer to devices and applications. If these edge-located announced resources are updated (e.g. by local applications or devices), those updates can be synchronized back to the central (IN-CSE) where the original resource resides. This ensures data consistency across the distributed IoT system.

7.2.3 Data Retrieval and Search Capability

oneM2M provides Discovery CSF as a set of features to allow efficient querying and retrieval of resources in the system. The oneM2M platform supporting MEC should support full or some of Discovery CSF. The Discovery CSF enables applications to search for resources in the system, filtering based on attributes, resource types, metadata, or specific criteria.

- The `RETRIEVE` Operation with `filterCriteria`: The core of the oneM2M discovery process is the use of the `RETRIEVE` operation on a target resource, combined with a `filterCriteria` parameter. Instead of retrieving the content of a single, known resource, the `RETRIEVE` operation in discovery mode is used to search for resources that match specific conditions within a defined scope. The `filterCriteria` parameter is a powerful tool that allows originators (the entities initiating the discovery request) to specify a wide range of search conditions. Some key parameters within `filterCriteria` include:
  - `resourceType` (`ty`): To search for resources of a specific type.
  - `labels` (`lbl`): Resources in oneM2M can be assigned arbitrary "`labels`" by their creators.
  - `resourceId` (`ri`): To search for resources by their unique identifier.
  - `creationTime` (`ct`) / `lastModifiedTime` (`lmt`): To search for resources created or modified within a certain time range.
- Advanced discovery (Semantic Discovery): As IoT systems grow in complexity and heterogeneity, simple attribute-based discovery can become insufficient. oneM2M addresses this with **semantic discovery**. This is an optional feature for the oneM2M platform to support MEC.

7.2.4 Registration Feature

In oneM2M, **registration** is the fundamental process by which an Application Entity (AE) or a Common Service Entity (CSE) makes itself known to another CSE to gain access to the oneM2M services and to be discoverable within the system. This process establishes a logical connection and creates corresponding resources on the Registrar CSE to represent the registered entity.

For MEC integration, the oneM2M platform should support two main types of registration:

- **AE Registration**: An Application Entity (AE) is a logical entity that represents an IoT application or a device application. Before an AE can utilize any oneM2M services (like creating containers for data, subscribing to events, or performing device management), it should register itself with a CSE.
- **CSE Registration**: A CSE (Common Service Entity) can also register with another CSE, typically a higher-level CSE in a hierarchical deployment. This forms a network of interconnected CSEs, allowing resources and requests to be forwarded across different CSEs. For example:
  - An ASN-CSE (Application Service Node CSE, typically on a constrained device or gateway) registers with an MN-CSE (Middle Node CSE, on a gateway or edge server).
  - An MN-CSE (e.g. IoT edge server running on a MEC platform) registers with an IN-CSE (Infrastructure Node CSE, in the cloud or core network).

7.2.5 Protocol Bindings

oneM2M is designed to be transport-agnostic, defining a common service layer with abstract primitives (e.g. CREATE, RETRIEVE, NOTIFY) and resources (e.g. <AE>, <container>, <contentInstance>) that can be mapped to diverse application-layer protocols. This mapping is called protocol binding in oneM2M. Protocol bindings specify how oneM2M operations and resource representations are encoded and transmitted over different protocols, enabling interoperability across heterogeneous IoT/MEC environments.

Key Bindings for MEC Integration are below:

- **HTTP:** Standard RESTful binding for cloud/edge interactions.
- **MQTT:** Pub/sub model for event-driven MEC Apps.
- **CoAP:** Lightweight binding for constrained devices (UDP-based).
- **WebSocket:** Full-duplex streaming for low-latency MEC data exchange.

To enable seamless integration between oneM2M and MEC, the oneM2M platform should support HTTP and MQTT as primary protocol bindings. These protocols are critical for ensuring interoperability, flexibility, and efficient data exchange in diverse MEC-IoT scenarios.

7.2.6 Required Features Summary

For seamless integration between MEC and oneM2M, the oneM2M platform should comprehensively support several key features. This includes real-time data management and efficient data exchange to ensure immediate processing at the edge. It also requires flexible data retrieval and search to quickly access relevant information. Integration is facilitated through standardized APIs, while standardized binding protocols (like HTTP, CoAP, MQTT, WebSocket) ensure versatile connectivity. Leveraging 5G network slicing and lightweight protocols, MEC applications can efficiently access oneM2M resources. This enables low-latency processing and real-time decision-making at the edge, maximizing the synergy between MEC and oneM2M.

Below is a summary of the key oneM2M features required for MEC interworking.

Table 7.2.6-1: Summary of key required features of oneM2M platform to support MEC

Key features	Description
Data and resource management	Enables immediate collection, processing, and storage of data at the edge. This requires comprehensive support for **oneM2M resources** (e.g. <container>, <contentInstance>) and their **CRUD operations** (CREATE, RETRIEVE, UPDATE, DELETE).
Real-time data exchange	Ensures fast and reliable flow of data between MEC applications and oneM2M resources. This is primarily facilitated by the **Subscription/Notification** mechanism and the use of **bi-directional announcement resources**.
Data retrieval and search	Allows MEC applications to quickly and precisely locate and query necessary data from vast edge datasets, heavily relying on the **discovery feature** with rich filterCriteria and **semantic discovery**.
Registration	Enables AEs and CSEs to make themselves known to the oneM2M system by creating specific resources (e.g. <AE>, <remoteCSE>). This is fundamental for identification, access control, and discoverability.
Protocol bindings	Supports various protocols (e.g. HTTP, CoAP, MQTT, WebSocket) for flexible and appropriate communication across diverse MEC environments, ensuring the mapping of oneM2M messages onto these transports.

Below is a summary of the key oneM2M resource types to support the selected features.

Table 7.2.6-2: Key resource types

Resource Types	Description
<AE> (Application Entity)	Represents an IoT application or a device-side application that interacts with the oneM2M system to use its services.
<CSEBase> / <remoteCSE> (Common Service Entity)	<CSEBase> is the root resource of a CSE, representing the oneM2M service platform itself. <remoteCSE> represents a remote CSE that has registered with the local CSE.
<container>	A generic resource used to organize and store instances of data, providing a structured way to manage information from devices or applications.
<contentInstance>	Contains the actual data payload or sensor readings, typically created as a child of a <container> resource.
<accessControlPolicy> (ACP)	Defines the access control rules and permissions for other oneM2M resources, governing who can perform which operations.
<subscription>	A resource created by a subscriber to monitor changes or events on a target resource, triggering a notification when specified criteria are met.
<AEAnnc> / <containerAnnc> / <remoteCSEAnnc> (Announcement)	Resources used to announce a local resource (AE, Container, or Remote CSE) to a remote CSE, making it discoverable and accessible across different CSEs.
<timeSeries>	A specialized resource type designed for efficiently storing and managing ordered time-series data entries, providing features for sampling and querying.

7.3 Use Case Selection Criteria

The discussion in this clause is fundamentally based on an edge computing compliant deployment, where oneM2M and MEC platforms are central to the architecture. Such an integration can consider several deployment options, each presenting distinct technical and business implications. In June 2023, oneM2M and ETSI MEC collaboratively developed a white paper titled "Enabling Multi-access Edge Computing in Internet-of-Things: how to deploy ETSI MEC and oneM2M" [i.16]. This white paper proposes four distinct deployment options for how oneM2M and MEC can be integrated and operated.

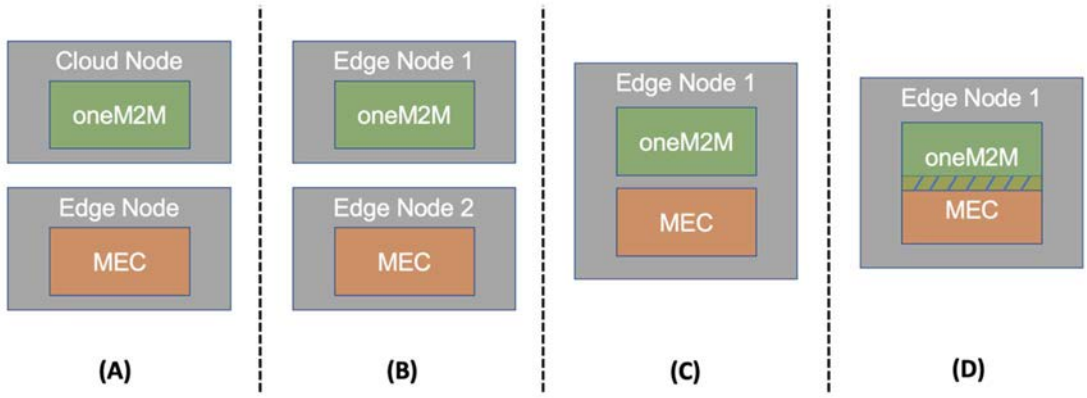

Figure 7.3-1: Deployment options integrating oneM2M and MEC with edge computing defined in ETSI white paper No. #59 [i.16]

Figure 7.3-1 (or 7.6-1 in the original white paper [i.16]) visually details these integration modes:

- **Option A: oneM2M as a Cloud, MEC as an Edge** This deployment scenario positions the IoT platform itself primarily on the cloud side, with MEC functioning at the edge. It represents one of the most common deployment configurations for cloud-based IoT platforms integrating with edge computing. While it allows for some benefits of edge computing through localized network and processing, it does not fully leverage the advantages of a 100 % edge computing environment, as the cloud remains the ultimate endpoint for data storage and management.

- **Option B: oneM2M and MEC as Edge Nodes on Different Physical Nodes** In this option, both oneM2M and MEC are deployed as edge nodes, but they reside on physically separate edge hardware. Compared to Option A, this setup allows for all data and information exchange between oneM2M and MEC to be performed locally at the edge, resulting in faster processing. Despite oneM2M IoT service providers and ETSI MEC entities potentially being distinct, this scenario is viable, particularly in the nascent stages of the edge computing market.

- **Option C: oneM2M and MEC on the Same Physical Edge Node** This scenario involves the oneM2M and MEC platforms being installed and operated on the very same physical edge node. This co-location can significantly enhance service performance by eliminating unnecessary data and information exchange between separate nodes. Implementing this option typically necessitates a Service Level Agreement (SLA) between the respective platform providers, and both platforms should support dynamic deployment capabilities across various edge nodes.

- **Option D: oneM2M and MEC Tightly Coupled within the Same Edge Node** This represents the deepest level of integration, where oneM2M and MEC platforms are physically coupled, often through direct API-level interworking. In this setup, the oneM2M platform can be recognized and function as an MEC application, fully leveraging all services and functions provided by the MEC environment to deliver a truly 100 % edge computing solution. Conversely, the MEC platform can directly offer data source, processing, and multi-access networking capabilities by hosting oneM2M as an integrated application. Achieving this level of tight coupling necessitates the development of standard documents defining clear interoperability and interworking specifications between the oneM2M and ETSI MEC platforms.

Use cases selected for an integrated oneM2M and MEC environment will fundamentally align with one of these four deployment options. Furthermore, these use cases should incorporate the relevant features, or subsets thereof, defined in clause 7.1 (MEC key features) and clause 7.2 (oneM2M features).

Specifically, the deployment strategy for core components such as oneM2M IN-CSE, oneM2M MN-CSE, oneM2M devices, and the MEC platform should correspond to one of the four options outlined in the aforementioned white paper, and each component should be configured to utilize the key features detailed in clause 7.

7.4 Assessment Criteria for Implementations

For both oneM2M and MEC platforms to be effectively utilized in Proof of Concept (PoC) deployments, they should adhere to specific implementation criteria. Fundamentally, these platforms need to be installed and configured according to one of the four deployment options defined in the collaborative white paper by oneM2M and ETSI MEC, as discussed in clause 7.3. Crucially, the chosen implementation should provide robust support for the core functionalities demanded by the specific use case selected for the PoC.

Therefore, it is a prerequisite that both the oneM2M and MEC platforms offer comprehensive support for the key features outlined in clause 7.1 (MEC key features) and clause 7.2 (oneM2M features), respectively. This full feature support ensures that the platforms are capable of enabling the complex interactions and data flows required for integrated edge computing scenarios.

Table 7.4-1 presents an analysis of available oneM2M implementations, such as ACME, tinyIoT, and Mobius, assessing their support for the essential features necessary to facilitate PoC engagements.

Similarly, Table 7.4-2 details the analytical results concerning available MEC Sandbox implementations, evaluating their capability to support the core features required for PoC deployments.

Table 7.4-1: Assessment criteria for oneM2M implementations

Selection Criteria	ACME	tinyIoT	Mobius
Resource management using CRUD operations	Supported	Supported	Supported
Resource discovery	Supported	Supported	Supported
Access Control Policy	Supported	Supported	Supported
Subscription/Notification	Supported	Supported	Supported
Bi-directional Announcement	Supported	Supported	No
<AE> registration	Supported	Supported	Supported
<CSE> registration	Supported	Supported	Supported
HTTP binding	Supported	Supported	Supported
MQTT binding	Supported	Supported	Supported

Selection Criteria	ACME	tinyIoT	Mobius
CoAP binding	Supported	Supported	Supported
WebSocket binding	Supported	Supported	Supported
Support resource type <AE>	Supported	Supported	Supported
Support resource type <CSEBase>	Supported	Supported	Supported
Support resource type <remoteCSE>	Supported	Supported	Supported
Support resource type <container>	Supported	Supported	Supported
Support resource type <contentInstance>	Supported	Supported	Supported
Support resource type <accessControlPolicy>	Supported	Supported	Supported
Support resource type <subscription>	Supported	Supported	Supported
Support resource type <AEAnnc> / <containerAnnc> / <remoteCSEAnnc>	Supported	Supported	No
Support resource type <timeSeries>	Supported	Supported	Supported

Table 7.4-2: Assessment criteria for MEC implementation

Selection Criteria	MEC Sandbox
IoT device and platform management	Supported
Real-time sensor data access	Supported
Service discovery and consumption	Supported
Network resource optimization	Supported
Application lifecycle management	Supported

8 Recommendations and Conclusions

In the preceding sections, specifically clause 7.4, a comprehensive analysis was conducted on available oneM2M and MEC implementations. This analysis assessed their capability to provide the essential standard features detailed in clause 7.1 (Required features for MEC) and clause 7.2 (Required features for oneM2M). Furthermore, the evaluation considered their suitability for supporting the use cases presented for Proof of Concept (PoC) development in clause 7.3.

The results of this assessment are conclusive: for MEC, the **MEC Sandbox** was found to provide all the necessary functionalities required for PoC implementations. Similarly, in the oneM2M domain, **ACME** and **tinyIoT** emerged as robust choices, confirming their full support for all features critical to PoCs. Notably, this includes their proficiency in enabling real-time data exchange, particularly related to bi-directional announcements between edge nodes and cloud IoT servers.

Therefore, based on this thorough analysis, it is strongly recommended that future PoC development, especially for use cases grounded in the four deployment options defined in the oneM2M/ETSI MEC white paper, leverage the **MEC Sandbox** in conjunction with **oneM2M ACME** and **tinyIoT**. These open-source implementations have demonstrated the necessary capabilities to effectively showcase the benefits and functionalities of integrated oneM2M and MEC solutions at the edge.

Annex A:
Change history

Date	Version	Information about changes
August 2025	4.1.1	Publication

History

Document history		
V4.1.1	August 2025	Publication