LF ENERGY

GEISA

GEISA Specification

Release 0.8.2-69bc3f7

Contributors to GEISA

Jun 26, 2026

Contents:

1 Abstract

The Grid Edge Interoperability & Security Alliance Specification defines an interoperable execution environment, application programming interface, and management interface for edge computing environments used by electric utilities. These edge computing environments are available on electric meters and distribution automation devices and enable a variety of customer-facing and utility-facing use cases.

2 Contributors

The following individuals contributed to the development of the Grid Edge Interoperability & Security Alliance specification. Our sincere thanks to them for the time and thought they put into this effort:

- Will Bell
- Matt Casperson
- Craig Cornwall
- Matt Gillmore
- Michael Garrison Stuber
- Bryan Green
- Ryan Houlette
- Marissa Hummon
- Don Jackson
- David Johnson
- Richard Lam
- Heather Lancaster
- Norm McEntire
- Kenny O'Dell
- Vlad Pambucol
- Jonah Petri
- Rick Steuer
- Brandon Thayer

- Scott Wegener

3 License

The Grid Edge Interoperability & Security Alliance specification is licensed under the Community Specification License (https://github.com/CommunitySpecification/Community_Specification/blob/main/1._Community_Specification_License-v1.md). This license governs all contributions, including, but not limited to, copyright and patent rights.

For convenience, the Community Specification License is provided below, in its entirety. Within this specification, the "Working Group" referenced in the license below is the Grid Edge Interoperability & Security Alliance, a Series of LF Projects, LLC.

Software developed under the auspices of Grid Edge Interoperability & Security Alliance is licensed under the Apache License.

3.1 Community Specification License 1.0

The Purpose of this License.

This License sets forth the terms under which:

1. The Contributor will participate in and contribute to the development of specifications, standards, best practices, guidelines, and other similar materials under this Working Group; and
2. The materials developed under this License may be used. It is not intended for source code. Capitalized terms are defined in the License's last section.

1. Copyright.

1.1. Copyright License.

Contributor grants everyone a non-sublicensable, perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as expressly stated in this License) copyright license, without any obligation for accounting, to reproduce, prepare derivative works of, publicly display, publicly perform, and distribute any materials it submits to the full extent of its copyright interest in those materials.

Contributor also acknowledges that the Working Group may exercise copyright rights in the Specification, including the rights to submit the Specification to another standards organization.

1.2. Copyright Attribution.

As a condition, anyone exercising this copyright license must include attribution to the Working Group in any derivative work based on materials developed by the Working Group. That attribution must include, at minimum, the material's name, version number, and source from where the materials were retrieved. Attribution is not required for implementations of the Specification.

2. Patents.

2.1. Patent License.

2.1.1. As a Result of Contributions.

2.1.1.1. As a Result of Contributions to Draft Specifications.

Contributor grants Licensee a non-sublicensable, perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as expressly stated in this License) license to its Necessary Claims in:

1. Contributor's Contributions; and
2. The Draft Specification that is within Scope as of the date of that Contribution,

in both cases for Licensee's Implementation of the Draft Specification, except for those patent claims excluded by Contributor under Section 3.

2.1.1.2. For Approved Specifications.

Contributor grants Licensee a non-sublicensable, perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as expressly stated in this License) license to its Necessary Claims included in the Approved Specification that are within Scope for Licensee's Implementation of the Approved Specification, except for those patent claims excluded by Contributor under Section 3.

2.1.2. Patent Grant from Licensee.

Licensee grants each other Licensee a non-sublicensable, perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as expressly stated in this License) license to its Necessary Claims for its Implementation, except for those patent claims excluded under Section 3.

2.1.3. Licensee Acceptance.

The patent grants set forth in Section 2.1 extend only to Licensees that have indicated their agreement to this License as follows:

2.1.3.1. Source Code Distributions.

For distribution in source code, by including this License in the root directory of the source code with the Implementation.

2.1.3.2. Non-Source Code Distributions.

For distribution in any form other than source code, by including this License in the documentation, legal notices, via notice in the software, and/or other written materials provided with the Implementation.

2.1.3.3. Via Notices.md.

By issuing a pull request or commit to the Specification repository's `Notices.md` file by the Implementer's authorized representative, including the Implementer's name, authorized individual and system identifier, and Specification version.

2.1.4. Defensive Termination.

If any Licensee files or maintains a claim in a court asserting that a Necessary Claim is infringed by an Implementation, any licenses granted under this License to that Licensee are immediately terminated unless:

1. The claim is directly in response to a claim against the Licensee regarding an Implementation; or
2. The claim was brought to enforce the terms of this License, including intervention in a third-party action by a Licensee.

2.1.5. Additional Conditions.

This License is not an assurance:

1. That any of Contributor's copyrights or issued patent claims cover an Implementation of the Specification or are enforceable; or
2. That an Implementation of the Specification would not infringe the intellectual property rights of any third party.

2.2. Patent Licensing Commitment.

In addition to the rights granted in Section 2.1, Contributor agrees to grant everyone a no-charge, royalty-free license on reasonable and non-discriminatory terms to Contributor's Necessary Claims that are within Scope for:

1. Implementations of a Draft Specification, where such license applies only to those Necessary Claims infringed by implementing Contributor's Contribution(s) included in that Draft Specification; and
2. Implementations of the Approved Specification.

This patent licensing commitment does not apply to those claims subject to Contributor's Exclusion Notice under Section 3.

2.3. Effect of Withdrawal.

Contributor may withdraw from the Working Group by issuing a pull request or commit providing notice of withdrawal to the Working Group repository's `Notices.md` file.

All of Contributor's existing commitments and obligations with respect to the Working Group up to the date of that withdrawal notice will remain in effect, but no new obligations will be incurred.

2.4. Binding Encumbrance.

This License is binding on any future owner, assignee, or party who has been given the right to enforce any Necessary Claims against third parties.

3. Patent Exclusion.

3.1. As a Result of Contributions.

Contributor may exclude Necessary Claims from its licensing commitments incurred under Section 2.1.1 by issuing an Exclusion Notice within 45 days of the date of that Contribution.

Contributor may not issue an Exclusion Notice for any material that has been included in a Draft Deliverable for more than 45 days prior to the date of that Contribution.

3.2. As a Result of a Draft Specification Becoming an Approved Specification.

Prior to the adoption of a Draft Specification as an Approved Specification, Contributor may exclude Necessary Claims from its licensing commitments under this Agreement by issuing an Exclusion Notice.

Contributor may not issue an Exclusion Notice for patents that were eligible to have been excluded pursuant to Section 3.1.

4. Source Code License.

Any source code developed by the Working Group is solely subject to the source code license included in the Working Group repository for that code. If no source code license is included, the source code will be subject to the MIT License.

5. No Other Rights.

Except as specifically set forth in this License, no other express or implied patent, trademark, copyright, or other rights are granted under this License, including by implication, waiver, or estoppel.

6. Antitrust Compliance.

Contributor acknowledges that it may compete with other participants in various lines of business and that it is therefore imperative that they and their respective representatives act in a manner that does not violate any applicable antitrust laws and regulations.

This License does not restrict any Contributor from engaging in similar specification development projects. Each Contributor may design, develop, manufacture, acquire, or market competitive deliverables, products, and services, and conduct its business in whatever way it chooses.

No Contributor is obligated to announce or market any products or services.

Without limiting the generality of the foregoing, the Contributors agree not to have any discussion relating to product pricing, methods or channels of product distribution, division of markets, allocation of customers, or any other topic that should not be discussed among competitors under the auspices of the Working Group.

7. Non-Circumvention.

Contributor agrees that it will not intentionally take or willfully assist any third party to take any action for the purpose of circumventing any obligations under this License.

8. Representations, Warranties and Disclaimers.

8.1. Representations, Warranties and Disclaimers.

Contributor and Licensee represent and warrant that:

1. Each is legally entitled to grant the rights set forth in this License; and
2. Neither will intentionally include any third-party materials in any Contribution unless those materials are available under terms that do not conflict with this License.

IN ALL OTHER RESPECTS, CONTRIBUTIONS ARE PROVIDED "AS IS." The entire risk as to implementing or otherwise using the Contribution or the Specification is assumed by the implementer and user.

Except as stated herein, CONTRIBUTOR AND LICENSEE EXPRESSLY DISCLAIM ANY WARRANTIES (EXPRESS, IMPLIED, OR OTHERWISE), INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, CONDITIONS OF QUALITY, OR TITLE, RELATED TO THE CONTRIBUTION OR THE SPECIFICATION.

IN NO EVENT WILL ANY PARTY BE LIABLE TO ANY OTHER PARTY FOR LOST PROFITS OR ANY FORM OF INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER FROM ANY CAUSE OF ACTION OF ANY KIND WITH RESPECT TO THIS AGREEMENT, WHETHER BASED ON BREACH OF CONTRACT, TORT (INCLUDING NEGLIGENCE), OR OTHERWISE, AND WHETHER OR NOT THE OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Any obligations regarding transfer, successors in interest, or assignment of Necessary Claims will be satisfied if Contributor or Licensee notifies the transferee or assignee of any patent that it knows contains Necessary Claims under this License.

Nothing in this License requires Contributor to undertake a patent search.

If Contributor is:

1. Employed by or acting on behalf of an employer;
2. Making a Contribution under the direction or control of a third party; or
3. Making the Contribution as a consultant, contractor, or under a similar relationship with a third party,

Contributor represents that they have been authorized by that party to enter into this License on its behalf.

8.2. Distribution Disclaimer.

Any distributions of technical information to third parties must include a notice materially similar to the following:

THESE MATERIALS ARE PROVIDED "AS IS." THE CONTRIBUTORS AND LICENSEES EXPRESSLY DISCLAIM ANY WARRANTIES (EXPRESS, IMPLIED, OR OTHERWISE), INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, OR TITLE, RELATED TO THE MATERIALS. THE ENTIRE RISK AS TO IMPLEMENTING OR OTHERWISE USING THE MATERIALS IS ASSUMED BY THE IMPLEMENTER AND USER.

IN NO EVENT WILL THE CONTRIBUTORS OR LICENSEES BE LIABLE TO ANY OTHER PARTY FOR LOST PROFITS OR ANY FORM OF INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER FROM ANY CAUSE OF ACTION OF ANY KIND WITH RESPECT TO THIS DELIVERABLE OR ITS GOVERNING AGREEMENT, WHETHER BASED ON BREACH OF CONTRACT, TORT (INCLUDING NEGLIGENCE), OR OTHERWISE, AND WHETHER OR NOT THE OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

9. Definitions.

9.1. Affiliate.

"Affiliate" means an entity that directly or indirectly Controls, is Controlled by, or is under common Control of that party.

9.2. Approved Specification.

"Approved Specification" means the final version and contents of any Draft Specification designated as an Approved Specification as set forth in the accompanying `Governance.md` file.

9.3. Contribution.

"Contribution" means any original work of authorship, including any modifications or additions to an existing work, that Contributor submits for inclusion in a Draft Specification and that is included in a Draft Specification or Approved Specification.

9.4. Contributor.

"Contributor" means any person or entity that has indicated acceptance of the License:

1. By making a Contribution to the Specification; or
2. By entering into the Community Specification Contributor License Agreement for the Specification.

Contributor includes its Affiliates, assigns, agents, and successors in interest.

9.5. Control.

"Control" means direct or indirect control of more than 50% of the voting power to elect directors of a corporation or, for any other entity, the power to direct management of that entity.

9.6. Draft Specification.

"Draft Specification" means all versions of the material (except an Approved Specification) developed by this Working Group for the purpose of creating, commenting on, revising, updating, modifying, or adding to any document that is to be considered for inclusion in the Approved Specification.

9.7. Exclusion Notice.

"Exclusion Notice" means a written notice made by issuing a pull request or commit to the repository's `Notices.md` file that identifies patents that Contributor is excluding from its patent licensing commitments under this License.

For issued patents and published applications, the Exclusion Notice must include the Draft Specification's name and the patent number(s) or title and application number(s), as applicable, for each issued patent or pending application being excluded.

If an issued patent or pending patent application that may contain Necessary Claims is not set forth in the Exclusion Notice, those Necessary Claims remain subject to the licensing commitments under this License.

For unpublished patent applications, the Exclusion Notice must provide either:

1. The text of the filed application; or
2. Identification of the specific part(s) of the Draft Specification whose implementation makes the excluded claim a Necessary Claim.

If option (2) is chosen, the effect of the exclusion is limited to the identified part(s) of the Draft Specification.

9.8. Implementation.

"Implementation" means making, using, selling, offering for sale, importing, or distributing any implementation of the Specification:

1. Only to the extent it implements the Specification; and
2. Only so long as all required portions of the Specification are implemented.

9.9. License.

"License" means this Community Specification License.

9.10. Licensee.

"Licensee" means any person or entity that has indicated acceptance of the License as set forth in Section 2.1.3.

Licensee includes its Affiliates, assigns, agents, and successors in interest.

9.11. Necessary Claims.

"Necessary Claims" are those patent claims, if any, that a party owns or controls, including claims later acquired, that are necessary to implement the required portions (including required elements

of optional portions) of the Specification that are described in detail and not merely referenced in the Specification.

9.12. Specification.

"Specification" means a Draft Specification or Approved Specification included in the Working Group repository subject to this License and the version of the Specification implemented by the Licensee.

9.13. Scope.

"Scope" has the meaning set forth in the accompanying `Scope.md` file included in this Specification's repository. Changes to Scope do not apply retroactively.

If no Scope is provided, each Contributor's Necessary Claims are limited to that Contributor's Contributions.

9.14. Working Group.

"Working Group" means this project to develop specifications, standards, best practices, guidelines, and other similar materials under this License.

4 Introduction

The Grid Edge Interoperability and Security Alliance (GEISA) Specification addresses interoperability for edge computing environments used by electric utilities. These edge environments are often found in Advanced Metering Infrastructure (AMI) meters, but may also be present in Distribution Automation (DA) devices, customer gateways, or other utility-owned devices.

The goal of the GEISA specification is to address end-to-end interoperability for edge environments.

Within the GEISA specification, interoperability is be broken down into four areas:

- Application and Device Mangement (ADM)
- Application Programming Interface (API)
- Linux Execution Environment (LEE)
- Virtual Execution Environment (VEE)

Vendor implementations of the GEISA specification may not support interoperability in all areas. For example, some vendors may support either a Linux execution environment (LEE), or a virtual execution environment (VEE), while other vendors may choose to support both, with a VEE running on top of or along side of a LEE environment. There is nothing in the specification that precludes supporting only one type of Execution Environment (LEE or VEE) or supporting both.

For clarity, this specification uses icons for each aspect of interoperability so it is clear which requirements apply to a given type of conformance. The icons are:

Throughout this specification there are blocks associated with each aspect of interoperability. In this introduction, the blocks explain, at a high level, the interoperability behaviors a given aspect supports. Blocks which discuss specific aspects of interoperability close with a GEISA pyramid icon ().

Application and Device Management (ADM) interoperability allows utilities to use a single management system to manage ADM-conformant devices from multiple vendors using an ADM-conformant management system. ADM defines the behaviors and protocols necessary to register and manage edge devices and the applications which run on them.

When this documentation notes something that is ADM-related, there may be an orchestrator's baton icon to highlight it.

While GEISA defines two specific execution environments (discussed below), ocassionaly there are requirements common to both.

When this specification discusses execution environments in general, there may

be a globe icon to highlight it.

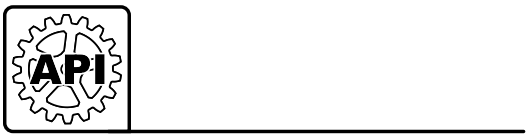
Linux Execution Environment interoperability allows software developers to have a consistent embedded Linux environment for running programs. LEE-conformant devices provide an isolated per-application execution environment containing a standard set of base libraries, a consistent filesystem layout, and other details which ensure application developers do not have to manage variations across different platform implementations.

When this specification notes something that is LEE related, there may be a Tux icon to highlight it.

Virtual Execution Environment (VEE) interoperability allows software developers to have a consistent virtual execution environment for running programs. VEE-conformant devices provide the

ability to run programs that rely on standard managed-code; that is, code executed under the supervision of a virtual machine that provides memory management, security, and isolation. The minimal supported languages for managed-code for GEISA VEE are C/C++ and Java, with their respective libraries.

When this specification notes something that is VEE related, there may be a connected cloud icon to highlight it.



Application Programming Interface interoperability allows software developers to interact with grid edge environment specific capabilities, such as meter readings, waveform data, sensors, and actuators in a consistent manner. GEISA uses a message-bus based API so that application developers are not constrained to a specific programming language.

When this specification notes something that is API related, there may be a gear icon to highlight it.

The goal of this specification is that two vendors working from the specification alone are able to create an implementation such that a GEISA-conformant application can run on nearly any GEISA-conformant platform without modification. Similarly, a vendor providing a GEISA-conformant ADM / management system should be able to manage a GEISA conformant platform without modifications.

This specification currently addresses:

- *Design Principles*
- *System Architecture*
- *Conformance*
- *Hardware Expectations*
- *Application & Device Management*
- *Linux Execution Environment*
- *Virtual Execution Environment*
- *Application Programming Interface (API)*
- *Security*

This specification follows [RFC2119] conventions, such as “MUST”, “MAY”, and “SHOULD” to indicate what is expected from a conformant implementation.

Please note that conformance for each of GEISA defined aspects of interoperability (ADM, API, LEE & VEE) is considered independently.

5 References

Note

Due to the way the PDF version of the specification is rendered, references are found in the Bibliography chapter. We hope to correct this in a future version of the specification.

6 Glossary

ADC
Analog-to-digital conversion Analog-to-digital converter

ADM
Application and Device Management

AMI
Advanced Metering Infrastructure

AII
Application Isolation Implementation

API
Applications Programming Interface

Application Certifier
An organization acting in this role that may independently assess, test, and/or countersign application artifacts for validation of behaviors prior to deployment. Optional in context of this version of the Specification.

Application Deployment Manifest
Operator-approved deployment manifest that defines effective application deployment parameters for an Application for a target environment.

Application Publisher
Organization or entity providing one or more Applications for Operator Testing, Deployment or Certification. Provides a signed package, vendor application manifest and associated collaterals.

Application Vendor
Role responsible for application production and publication responsibilities in a GEISA workflow.

CoAP

Constrained Applications Protocol

COE

Common Operating Environment

The COE is the software platform specification which defines the software components, interfaces, and processes needed to implement a GEISA-conformant platform. The COE is designed to be flexible and modular, allowing for a variety of implementations that can meet the needs of different use cases and environmental constraints. The COE is intended to provide a common foundation for GEISA implementations, while also allowing for customization and innovation by implementers.

DER

Distributed Energy Resource

Distributed Energy Resource is a generic term that typically includes photo-voltaics coupled with a smart inverter, battery energy storage systems, electric vehicles, demand response, and other dispatchable load or generation available on the electric distribution network. DERs are typically owned by consumers or businesses, rather than utilities or wholesale energy market participants.

DTLS

Data Transport Layer Security

FAN

Field Area Network, generally owned/operated by the Utility or Operator. Sometimes referred to as AMI network.

EE

Execution Environment - one of LEE (Linux Execution Environment) or VEE (Virtual Execution Environment) which Edge Applications run on.

EMA

Edge Management Agent

The EMA is the logical component of a GEISA ADM conformant platform which implements ADM capabilities on the edge device(s).

EMS

Edge Management System - the overall system implementing the set of capabilities of the ADM Pillar, which will include the LwM2M server and client typically but may also include other software and/or capabilities.

Note: with the electric utility industry, EMS often means *Energy* Management System. Similarly, within the network management system EMS often means *Element* Management System. Within this specification EMS is used exclusively for Edge Management System.

Edge Application

Workload running within a GEISA execution environment using GEISA platform services and API interfaces.

GEISA
: Grid Edge Interoperability and Security Alliance

GPIO
: General Purpose Input/Output

GUI
: Graphical User Interface

HAN
: Home Area Network

Hybrid Application Model
: A hybrid application model in context of GEISA refers to an application which may use a combination of edge/local and cloud resources for determination of actual conditions, or additional upstream analysis on the aggregation of edge-determined conditions. For example, a specific edge application may determine some specific conditions locally, but also share data upstream to a cloud-based application to perform additional analysis to determine if other conditions may exist, or to determine if the locally-determined conditions are part of a larger pattern of conditions.

LAN
: Local Area Network

LEE
: Linux Execution Environment

LwM2M
: Lightweight Machine-to-Machine

MQTT
: MQTT is a light-weight publish and subscribe protocol formerly known as Message Queueing Telemetry Transport.

OS
: Operating System

Platform Provider
: Role responsible for supplying platform hardware, platform software, and associated platform trust material where applicable.

PKI
: Public Key Infrastructure

Platform Implementation
: A combination of hardware and software which provides one or more GEISA conformant interfaces (ADM, API, EE).

POSIX
: Portable Operating System Interface

RMS
: Root Mean Square

SPI
: Serial Peripheral Interface

System Operator
: Role with deployment-environment authority for approving, configuring, authorizing, deploying, and operating GEISA-managed devices and applications.

Vendor Application Manifest
: Publisher-supplied application manifest provided with an application artifact as input to operator deployment decisions.

userid
: In context of GEISA MQTT APIs, this is a platform-local unique identifier assigned to a deployed GEISA application instance on that local device. It is used for topic routing and message correlation within a device. Implementations may map this identifier to underlying platform or application constructs such as application manifest details, container identities, or other information in order to ensure local uniqueness.

  The userid is unique within the scope of a device but is not required to be globally unique (as of this specification version) and does not represent a human user.

  In this version of the GEISA specification, platforms are not required to support concurrent execution of multiple instances of the same application version on a single device. Future revisions may expand this behavior.

VEE
: Virtual Execution Environment

7 Design Principles

The Grid Edge Interoperability & Security Alliance specification is written for constrained resource embedded industrial devices. A constrained embedded industrial device may consist of a simple microcontroller (e.g. Cortex-M class core), a more complex microprocessor (e.g. Cortex-A class core), or a heterogeneous or modular environment with both.

These devices are expected to operate in the field without manual intervention for prolonged periods of time; often two decades or longer. While these devices have network interfaces and sensors (see *Hardware Expectations*) they do not typically have on-board (graphical) user interfaces.

This device target informs the overall design principles of the specification.

GEISA design principles include:

- Interoperability
- Constrained Environment
- Minimal Implementation
- Core Specification with Extensions
- Security

Each design principle is described in more detail below.

7.1 Interoperability

As discussed in the *Introduction*, GEISA defines four types of interoperability:

LEE

Linux Execution Environment **LEE** interoperability  provides a consistent operating system environment built using Linux. The Linux Execution Environment provides a well-known open development environment including many standard libraries and makes it easy to integrate additional technology.

Applications written for the GEISA LEE will run on any LEE-conformant device.

VEE

Virtual Execution Environment

VEE interoperability provides a consistent virtual execution environment. The Virtual Execution Environment provides a fully-isolated managed code environment which supports the widely-used C/C++ and Java ® language specifications.

Applications written for the GEISA VEE will run on any VEE-conformant device.

ADM

Application and Device Management ADM interoperability enables conformant devices to interoperate with conformant management systems.

APIs

Application Programming Interface API interoperability provides consistent access to resources such as sensors, actuators, networking, and more.

Note that GEISA does not currently provide a tool-chain or a base-platform implementation.[1]

This release of the GEISA specification defines **source-code** interoperability for the LEE, VEE, & API, and network interoperability for ADM.

Source written for the GEISA LEE should be compilable without modification, and without needing conditional compilation or platform specific directives. Similarly, source written for the GEISA VEE should be compilable for the target virtual execution environment with needing conditional compilation or platform-specific directives.

As an example, shown below is a "do-nothing" C source code program that should compile using any GEISA LEE toolchain, producing an executable that runs on any GEISA LEE.

```c
#include <stdio.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
     while(1) {
         sleep(60); // sleep for 60 seconds
     }
     return 0;
 }
```

As another example, shown below is a "do-nothing" Java source code program that should compile using any GEISA VEE toolchain, producing an executable that runs on any GEISA VEE.

[1] GEISA may provide a toolchain and base implementation in the future if there is interest and support from the GEISA community.


```java
public class GeisaHello {
   public static void main(String[] args) {
       while (true) {
           try {
               Thread.sleep(60000); // sleep for 60 seconds (60,000␣
↪milliseconds)
           } catch (InterruptedException e) {
               e.printStackTrace();
           }
       }
   }
}
```

Warning

The LEE and VEE are two different environments. The LEE is specifically a Linux environment.

The VEE is agnostic to the underlying OS/RTOS, providing source-level interoperability through supporting well-known languages.

It is possible that a platform may support both LEE and VEE. In this case, an application developer may need to determine the proper methods that allows it to run on one or both execution environments when properly packaged. System operators may need to select applications according to the provided EE.

See *System Architecture* for further discussion.

7.2 Constrained Environment

The GEISA EE is a minimal resource environment, having constraints on CPU, RAM, storage, and networking. See *Hardware Expectations* for specific minimums that the GEISA specification assumes.

As much as possible, resources should be reserved for GEISA applications, instead of being consumed by the underlying operating system. While a GEISA conformant EE may offer more, it MUST reserve the following MINIMUM resources available for GEISA applications:

- For GEISA LEE  :
  - **256MB of RAM**
  - **256MB of persistent storage**
- For GEISA VEE  :
  - **1 MB of RAM**

- **8 MB of Flash memory** for code storage and execution
- **8 MB of persistent storage**

Additionally, although hardware performance will vary considerably between platforms, GEISA EE SHOULD provide a minimum of **50% of the CPU** for GEISA applications .

Efficiency is critical. The GEISA EE shall provide only those services which are so widely required that it would be less efficient to *not* provide them.

7.3 Minimal Implementation

Keeping systems up-to-date is challenging, both for utilities and for vendors. The GEISA EE favors the minimal implementation; that is, the less there is, the less there is to maintain and the less there is to attack.

If there is an option that is not needed, then turn it off. If there is an unwanted or un-necessary feature, leave it out. GEISA should include only what is required.

7.4 Core Specification with Extensions

In addition to the four keys aspects of interoperability (ADM, API, LEE, and VEE), future extensions may be added to to the GEISA specification may be added as the community determines the need.

Future extensions may define new areas of interoperability conformance beyond ADM, API, LEE, and VEE, or they may enable new capabilities that are only needed by selected devices types or in specific markets. Extensions will allow GEISA to retain its *Minimal Implementation* design principle, while still allowing it to grow to meet the needs of platform vendors, application developers, and system operators.

7.5 Security

Security is equally as important as interoperability within the GEISA specification. All protocols included in the GEISA specification support robust security. Security details and requirements are provided throughout the specification. At every level, from minimizing the attack space, to hardening of the APIs and all services, GEISA security is foundational.

8 Operations

This chapter provides a high-level operational view of how GEISA roles, systems, artifacts, and interoperability areas relate to each other end to end.

It is intended to help readers understand the overall system context before moving into the more detailed materials covered elsewhere such as *System Architecture*, *Application & Device Management*, *Linux Execution Environment*, *Virtual Execution Environment*, *Application Programming Interface (API)*, and *Security*.

Note

This chapter is explanatory in nature. Detailed protocols, APIs, payloads, lifecycle, and security requirements are defined in the relevant chapters of this specification.

8.1 Purpose and Scope

GEISA is intended to support end-to-end interoperability across the lifecycle of utility edge platforms, edge applications, and other upstream systems. In practice, the overall system can be difficult to understand when viewed only through one protocol, one component, or one's own specific organizational responsibilities.

This chapter reframes the system in terms of GEISA roles and major operational interactions. It is intended to clarify how the management plane, execution environments, application interfaces, trust decisions, and operational reporting concerns all fit together at a system level.

The operational view in this chapter complements the interoperability goals described in *Introduction* and the general architectural material in *System Architecture*.

8.2 Roles and Authorities

GEISA describes operational interactions in terms of roles rather than broad organization categories, as a single organization may perform one or more roles depending on the workflow being discussed and specific implementations.

For example, a device manufacturer may also provide managed services and, in a particular deployment, may act in the role of Platform Provider, EMS Provider, or System Operator. In GEISA, responsibilities and decision authority are tied to the role being performed in the relevant interaction, regardless of the type of organization with specific role responsibilities. However, there is one notable callout: the utility is expected to often be the one driving the underlying operational decisions, even in cases where another organization is performing the day-to-day operational duties on their behalf.

The principal roles used in this operational view are:

- *System Operator*

  The role responsible for approving, configuring, authorizing, deploying, and operating GEISA-managed devices and applications within a specific operating environment in accordance with agreed-on policies. The Operator effectively runs the day-to-day systems, although they may be beholden to another organization such as the Utility to dictate actual policies to be followed.

- *Platform Provider*

  The role responsible for supplying the platform hardware, platform software, and associated platform trust material where applicable. This may be a single provider or a Hardware Provider may be separate. In terms of GEISA, Platform Provider generally is noting the provider of the system software that is GEISA-conformant.

  - *Application Publisher* / *Application Vendor*

  The role responsible for producing and signing an application artifact and supplying its vendor manifest and associated publisher trust material.

- *Application Certifier*

  An optional (as of this version of the specification) role that may independently assess, test, or countersign application artifacts as validated and safe for deployment purposes. Depending on the applicable program or market, this assessment may emphasize either a “does no harm” model focused on behavioral and policy safety, a more detailed “does what is promised” model focused on functional or analytic claims, or both.

- *Edge Application*

  A workload running within a GEISA execution environment and using GEISA platform services and the GEISA *Application Programming Interface (API)*.

- ADM / Application and Device Management

  ADM is the GEISA pillar for *Application & Device Management*. It spans the capabilities

required for application ingestion, operator approval for deployment, deployment, activation, deactivation, updates, and broader lifecycle management of both the platform and the applications running on it. These capabilities may be realized by a single EMS that leverages LwM2M, or by multiple cooperating components, provided the required GEISA ADM behaviors and transactions are satisfied. On the edge device, this capability set includes the EMA and related edge-side management functions.

- *EMS*

  The management system role responsible for exposing GEISA-conformant management behavior to the operator and communicating with conformant platforms as described in *Application & Device Management*.

- *EMA*

  The platform-side management function that implements the required ADM support on a GEISA platform on edge devices.

The following terms are also important to this chapter:

- *ADM* and *API* are GEISA interoperability areas and components covered in detail in *Application & Device Management* and *Application Programming Interface (API)*.
- *LEE* and *VEE* are GEISA execution environments covered in detail in *Linux Execution Environment* and *Virtual Execution Environment*.
- *Platform Implementation* describes the realized combination of hardware and software that provides one or more GEISA-conformant interfaces.

When policy, approval, deployment scope, resource allocation, or trust authorization is in question, the System Operator has final authority for the target deployment environment.

8.3 End-to-End Operational Context

At a high level, the System Operator uses an *EMS* to manage conformant platforms, authorize application deployment, and obtain operational visibility into the deployed fleet.

Within the platform, *ADM* provides the management behavior needed for device onboarding, lifecycle control, and application deployment as described in *Application & Device Management*. The platform may provide one or both of the GEISA execution environments, *LEE* and *VEE*, as described in *Linux Execution Environment* and *Virtual Execution Environment*. Applications use the GEISA *API* to interact with platform capabilities and platform-provided data as described in *Application Programming Interface (API)*.

The Application Publisher supplies the application artifact and vendor manifest. The System Operator determines whether that application is approved for use in the relevant environment, such as test, limited deployment, or production deployment. The EMS then carries out the authorized deployment and lifecycle actions on the target devices.

The Platform Provider supplies the platform implementation and associated device or platform trust material where applicable. The same provider may also supply additional services or systems, but those services are distinct from the role boundaries used within this specification.

This figure is intentionally high-level. It is provided to help readers understand how the principal GEISA roles, systems, and interoperability areas relate end to end.

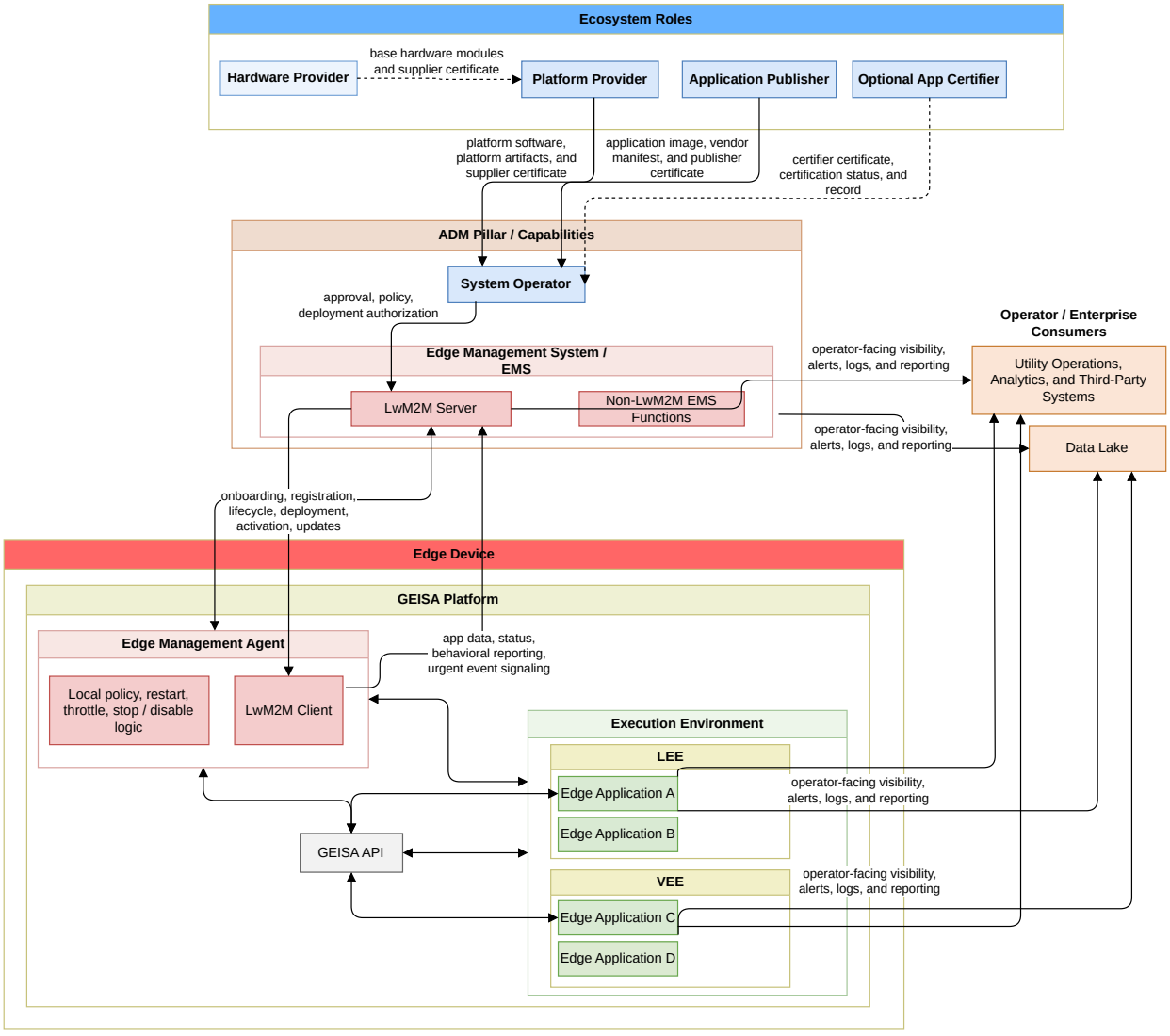


Figure 8.1: High-level GEISA operational context showing how principal roles, platform-side and off-device management functions, execution environments, and utility or enterprise interaction points relate end to end (2 diagrams, second is a continuation from System Operator through to to the Edge Device)

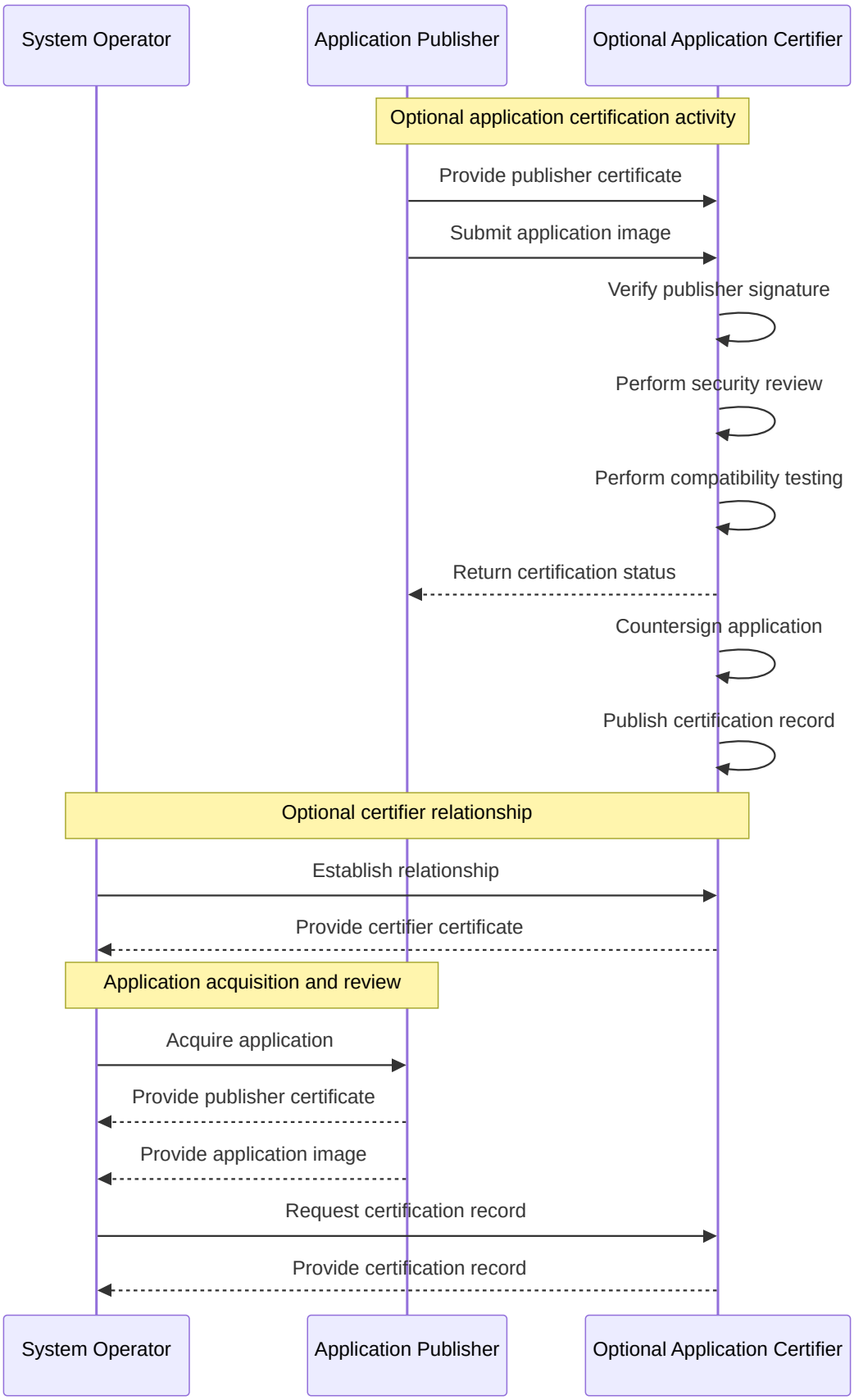


Figure 8.2: High-Level End to End Sequence Diagram 1 of 2

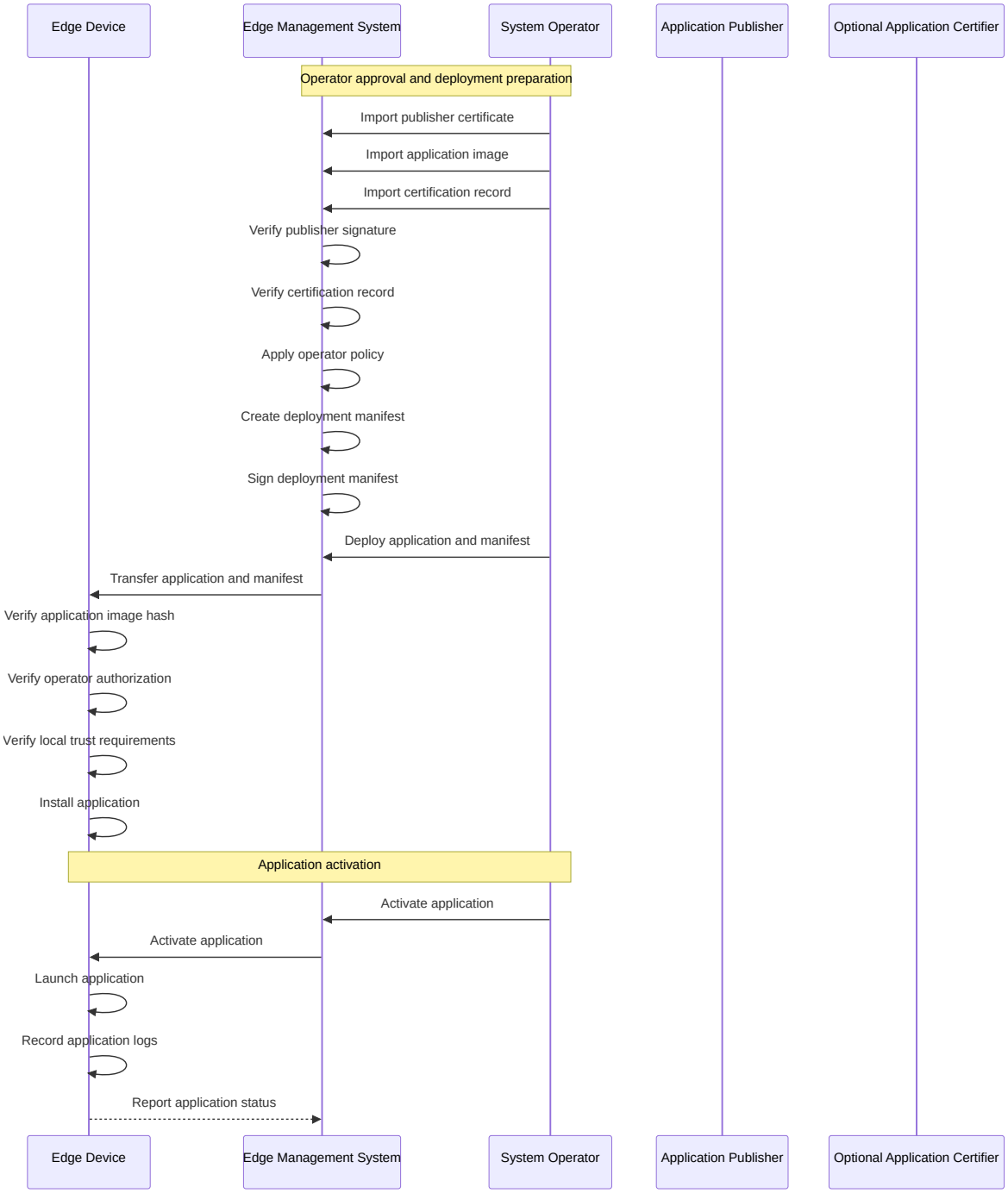


Figure 8.3: High-Level End to End Sequence Diagram 2 of 2

8.4 Operational Capability

This chapter is explanatory and does not add or alter specific requirements detailed elsewhere in this specification. In particular, *Application & Device Management* continues to define required interoperable ADM behaviors and transactions.

Operational capability includes management, lifecycle, deployment, visibility, reporting, and integration concerns across platform and operator workflows.

8.5 Device Onboarding and Management Overview

GEISA device onboarding establishes the management relationship between a conformant platform and a conformant EMS. In operational terms, this includes device provisioning, network attachment, bootstrap or initial trust establishment, registration, and subsequent lifecycle management.

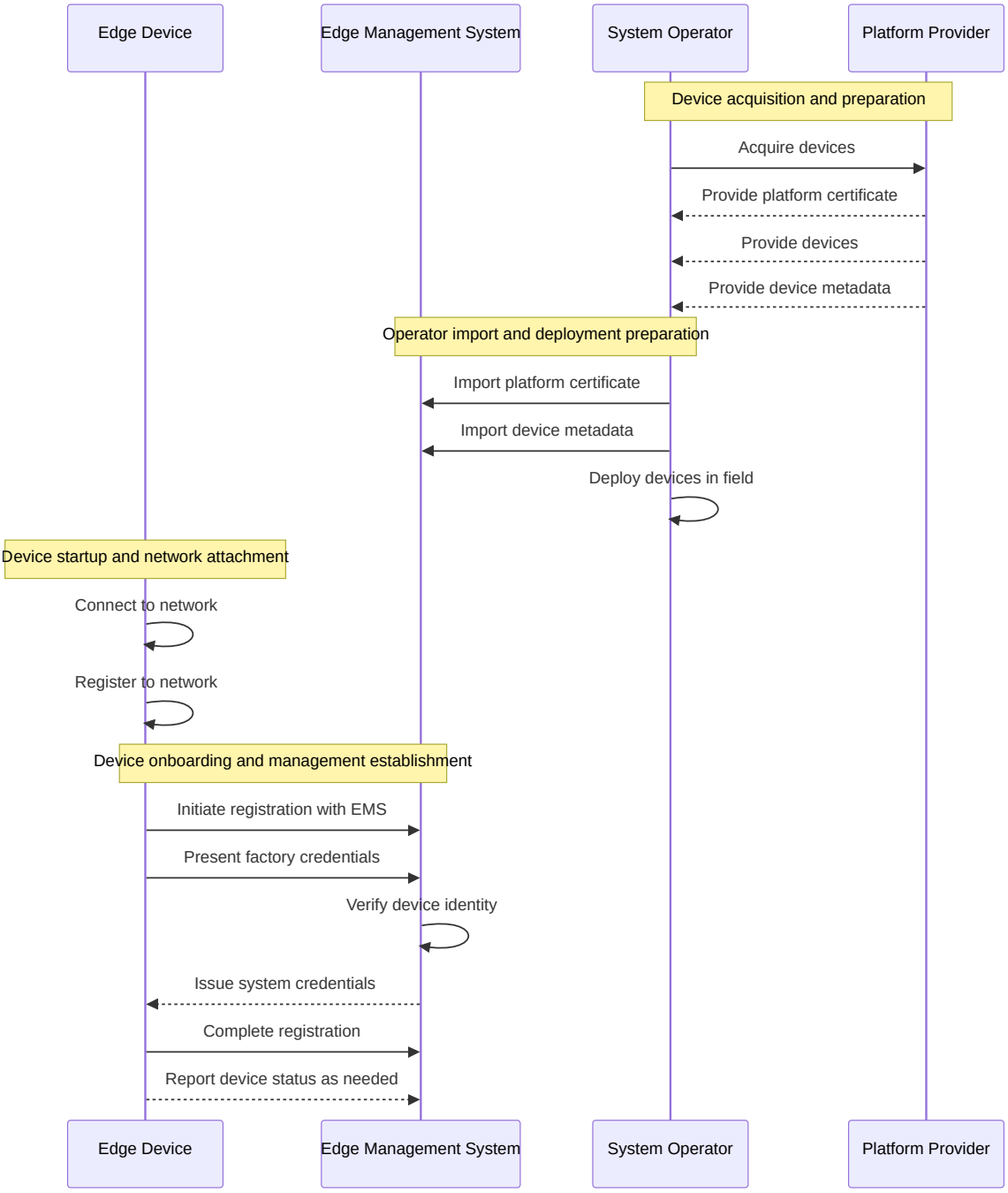


Figure 8.4: Reference device onboarding and management establishment flow retained for comparison and discussion.

Additional detailed device onboarding workflows and requirements may be added in a future revision of the specification.

8.6 Application Approval and Deployment Overview

GEISA application deployment begins with application sourcing and review, continues through operator authorization and deployment preparation, and ends with transfer and installation on target devices.

This specification distinguishes between the vendor manifest supplied by the Application Publisher and the effective deployment manifest approved by the System Operator. Operationally, this means publisher-provided identity, compatibility, defaults, and requested resources are inputs to the deployment decision, but the final approved deployment parameters remain under operator control.

Additional detailed application approval and deployment workflows and requirements may be added in a future revision of the specification.

8.7 Off-Device Communication Approval

Applications may request message-based communication, IP socket communication, or both. Operationally, these requests are inputs to the deployment decision and do not entitle an Application to unmanaged network access. The System Operator determines whether requested communication is approved for the target deployment environment.

Direct IP socket communication should be reviewed as a distinct operational and security decision. Operators may choose to deny direct IP communication by default, approve it only for selected applications or destination classes, restrict it to operator-controlled endpoints, or require the platform to mediate approved traffic through routing, NAT, firewalling, forwarding, transparent proxying, or shared upstream connections.

The approval review should consider endpoint ownership, protocol and port scope, certificate and trust behavior, expected data volume, retry behavior, outage behavior, inbound exposure, fleet network capacity, and operational visibility. Inbound IP socket communication generally presents higher operational and security risk than outbound-initiated communication and may be limited to local endpoint use cases or denied by operator policy. This version of the specification does not define interoperable support for application-hosted listeners reachable from upstream operator networks or for peer-to-peer application communication over the AMI network. Those use cases may be considered in a future version if driven by operator or application requirements.

Platform-originated reporting should provide timely operator visibility into denied access attempts, quota limits reached, unexpected egress attempts, proxy or forwarding failures, and other communication policy violations. The reporting mechanism may be defined by ADM/EMS behavior, platform event reporting, event logs, app accounting, or future GEISA operational event extensions.

8.8 Application Activation and Runtime Visibility

After installation, GEISA applications are activated under operator control and execute within the platform's selected execution environment. During runtime, the platform and EMS provide visibility into application and platform state, including activation state, execution state, status, and other operationally relevant data.

Additional detailed application activation and state information requirements may be added in a future revision of the specification.

8.9 Operational Reporting and Visibility

GEISA deployments require operational visibility into both the managed platform and the applications running on it. For purposes of this operational discussion, it is useful to distinguish between four broad reporting classes:

- Application alerts

  Near-real-time when feasible. These are application-originated alerts or condition notifications for which delayed awareness may materially reduce operator usefulness.

- Application data

  Data generated by applications for their own systemic use and consumption. This data is not necessarily as time-critical as application alerts and may be pushed on a periodic schedule or pulled periodically in a manner agreed to by the utility, relevant platform provider, and application publisher.

- Platform-originated behavioral events

  Near-real-time when urgent, otherwise routine or batched as appropriate. These are platform-originated observations or actions relating to application or platform behavior, such as repeated restart failure, policy violations, throttling, disablement, denied access attempts, unexpected outbound messaging, signature or hash mismatch events, or severe and persistent threshold violations.

- Routine operational reporting

  Slower and/or batched as appropriate. This includes lower-urgency operational data intended primarily for visibility, trending, support, audits, planning, license or usage review, and similar operational purposes.

This model preserves an important distinction between what an application reports, what the platform reports about the application, and what the platform or operator may do as a result of that behavior and data. Application-originated condition notifications or data messages may flow through GEISA messaging and off-device communication paths, but those messages are distinct from the broader operational reporting model for the platform or managed device fleet.

Expected categories of operational reporting and visibility include:

- Lifecycle

  Install counts, versions, uninstall counts, start and stop counts, and stated or desired reasons for transitions. This information is mostly routine, though some lifecycle transitions may require urgent visibility.

- Behavioral / Stability

  Crashes, forced shutdowns, restarts, throttling events, crash logs, policy violations, signature or hash mismatches, denied access attempts, unexpected outbound messaging, and similar platform-originated behavioral events. This category includes both routine visibility and urgent events that may require operator intervention.

- Resource Consumption

  Resource usage by application instance or group, threshold crossings, average and peak CPU, time above threshold, average and peak memory, and similar bounded-behavior metrics. Resource reporting is generally routine, but threshold crossings may escalate in urgency based on severity or persistence.

- Platform / API / Data Reliability

  Platform or device errors, API failures, failed calls, data gaps, timeout rates, and related indications that the deployed application or platform capability is not performing as intended. These conditions may be urgent when they materially impair the intended operational purpose of an application or platform capability.

- Connectivity and Environment

  Connectivity uptime, time synchronization status or drift, failed sends, average latencies, DNS failures, storage wear warnings, and similar conditions affecting reliable operation. This category is especially important when applications are deployed for emergency, safety, or other time-sensitive operational visibility.

- License Count / Usage

  Install counts, usage counts, or similar license-related visibility as applicable to the deployment or application business model. This information is generally routine and suitable for slower or batched reporting.

Detailed operational reporting and visibility workflow material may be added in a future revision of the GEISA specification.

8.10 Utility and Enterprise Interaction Points

GEISA exists within a broader utility operational environment. In practice, the operator-facing GEISA systems and workflows may interact with utility or enterprise systems responsible for fleet operations, metering operations, analytics, customer or service workflows, and other business processes.

Detailed utility and enterprise integration workflow material may be added in a future revision of this chapter.

8.11 Application Certification

This chapter distinguishes between two broad certification models that may be relevant to GEISA application ecosystems. The industry continues to evolve and this may be expanded in a future revision. Costs as well as capabilities of individual providers able to be located to perform this function as a Utility may desire may also come into play.

Regardless, there are two broad types of certification we will consider at this time.

- Basic - ‘does no harm’ certification

  This model focuses primarily on whether the application remains within approved behavioral, resource, security, and policy bounds for safe deployment. Examples may include validating that the application does not access interfaces or data it was not granted, does not exhibit unacceptable restart or memory-growth behavior, does not exceed approved policy or threshold limits, does not produce unexpected outbound messaging, and does not otherwise destabilize or negatively impact the platform or adjacent functions.

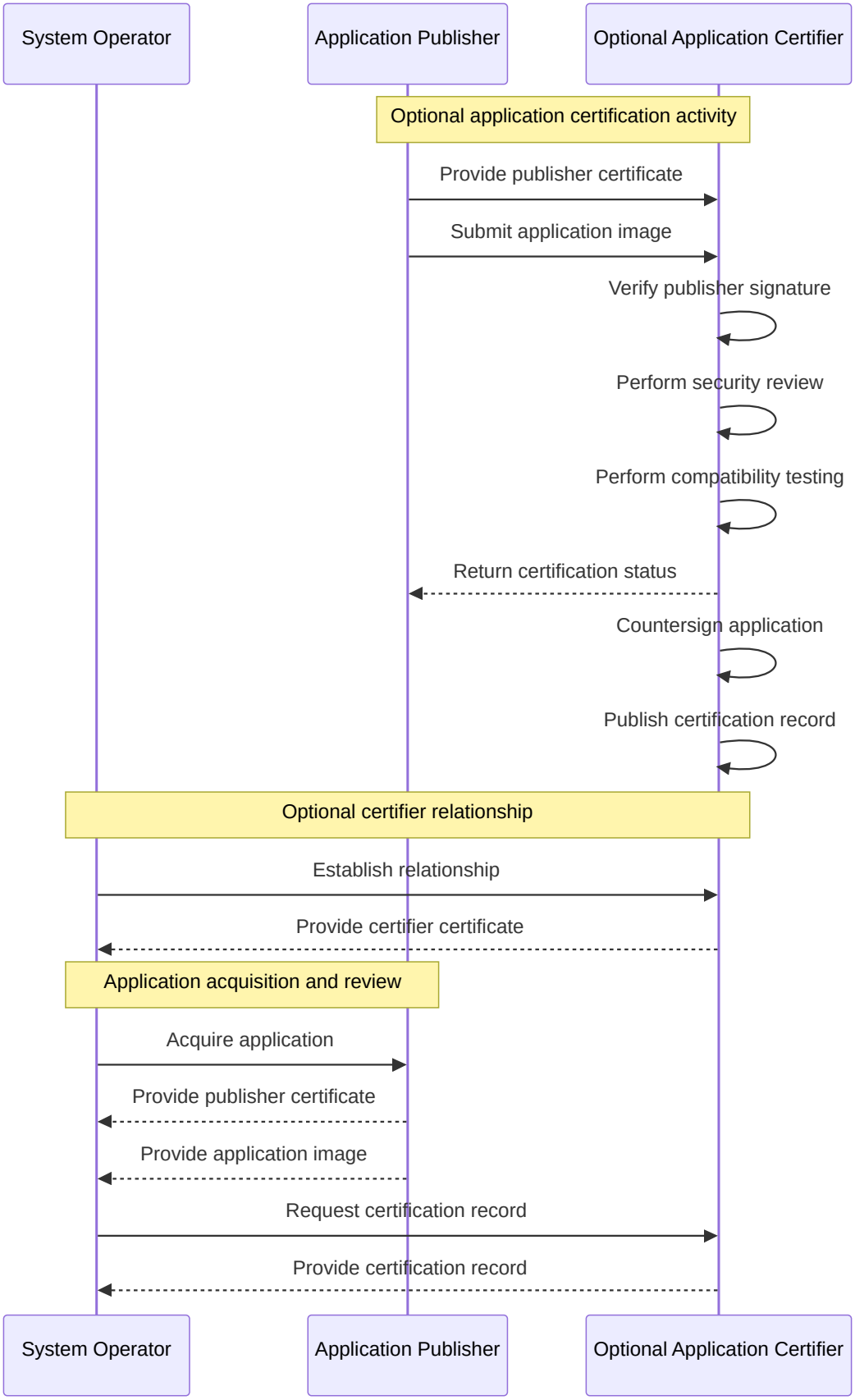
  Evidence of interest to such a program may include denied access attempts, restart storms, persistent memory or resource threshold violations, throttling, shutdown or disablement actions, signature or hash mismatch events, and related Platform-originated behavioral events.

- Deeper Functional Certification

  This model focuses more directly on validating whether an application satisfies a deeper specific functional and/or performance claim, such as analytics accuracy, condition-detection fidelity, expected true or false positive rates, interoperability with specific external devices, or similar claims. It is essentially - ‘does this application do what is promised?’

  This type of certification may be substantially more complex and expensive, particularly for applications that make use of local AI inferencing, hybrid edge/cloud processing, behind-the-meter integrations, or other advanced workflows.

  This model may also require access to details, methods, or validation detail that some publishers may consider sensitive or proprietary.


System Operator
Application Publisher
Optional Application Certifier
Optional application certification activity
Provide publisher certificate
Submit application image
Verify publisher signature
Perform security review
Perform compatibility testing
Return certification status
Countersign application
Publish certification record
Optional certifier relationship
Establish relationship
Provide certifier certificate
Application acquisition and review
Acquire application
Provide publisher certificate
Provide application image
Request certification record
Provide certification record
System Operator
Application Publisher
Optional Application Certifier

As of this version of the specification, GEISA does not define a singular or specific certification program or require either model. However, it is useful to distinguish these models explicitly because they address different operator, certifier, publisher, and ecosystem concerns. In the near term, GEISA or GEISA-conformant systems may more naturally support does-no-harm style certification than exhaustive functional-claim certification, although Platform Providers and Operators should consider if a given Platform provides enough capabilities to support both types of Certification.

8.12 Future Considerations

This chapter is intended to establish the operational frame for the GEISA system. Additional operational detail is expected to be added over time.

Examples of areas that may be expanded in a subsequent version of the specification include detailed application certification workflows, richer operator-side discovery or notification of newly available applications, enterprise integration patterns, meter-specific lifecycle workflows such as meter swap or move-in and move-out support, more detailed operational reporting and visibility models, and more explicit event taxonomies for application alerts and Platform-originated behavioral events.

9 System Architecture

9.1 General Architecture

As discussed in *Introduction*, the Grid Edge Interoperability & Security Alliance specification describes four types of interoperability: ADM, API, LEE, and VEE. These types of interoperability live in a general system context, which is shown below in Figure 9.1.

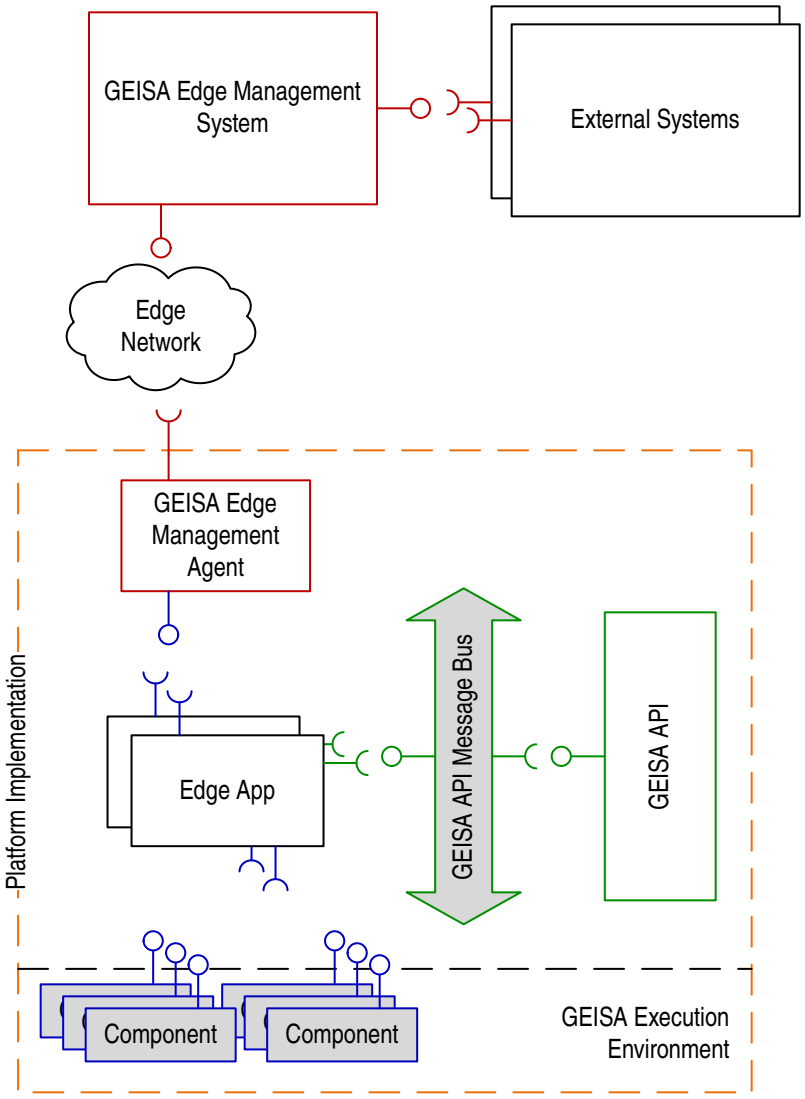


Figure 9.1: GEISA Architecture

GEISA is focused on interoperability and security. Where it is possible to maintain interoperability without specifying aspects of the implementation, details are left to the implementer.

In a full implementation of the GEISA architecture, there is a GEISA Edge Management System (EMS) that implements the ADM interface as described in *Application & Device Management*. The EMS implements a network protocol based interface and is able to use that protocol to interact with ADM interface conformant GEISA platform implementations. Internal details of the EMS

are beyond the scope of this specification. GEISA simply requires that a conformant EMS expose the required functions to the system operator in some meaningful way, and that it implements the specified network protocol and transactions in response to user commands. An EMS could provide a GUI, a command line interface, or an API and still be considered to be conformant.

Interacting with the GEISA EMS is one or more platform implementations. An ADM-conformant platform interface will be able to communicate with the GEISA EMS using the ADM protocol and transactions. In Figure 9.1 above, the ADM conformant implementation is shown as the GEISA Edge Management Agent. Implementation details are beyond the scope of this specification. Implementers may provide a standalone EMA within their platform, or integrate EMA functions within other elements of their platform as they see fit.

GEISA ADM elements are shown in red in Figure 9.1 above.

The system operator is able to use the GEISA EMS to deploy and activate GEISA edge applications to GEISA platform implementations. A fully conformant GEISA platform implementation will provide both a GEISA-conformant EE / execution environment and a GEISA-conformant API / application programming interface.

GEISA defines two different execution environments:

- **Linux Execution Environment (LEE)**
- **Virtual Execution Environment (VEE)**

Platforms may offer one or both of these execution environments (EE).

While the GEISA API should be accessible from both EEs, there is no expectation that code written for one EE will operate on the other without being ported.

Note

It is possible, though not planned at this time, that GEISA may support additional execution environments in the future.

LEE GEISA applications are provided as container images, as detailed in *Application Isolation*. An LEE conformant GEISA implementation is able to mount a container image and grant that container the rights the system operator permitted the application in the deployment manifest. The container

runtime used by the platform implementation is out of scope, but conformant implementations MUST be able to enforce the permissions and controls the GEISA specification requires.

An LEE-conformant GEISA implementation must provide a set of *Linux Base Libraries* and *Core Services* to containers running in the environment to minimize the size of each provided application container image. The goal of GEISA LEE conformance is to provide a consistent and efficient execution environment to edge applications running on the platform. Platform implementers may use any GNU/Linux variant they see fit, provided it meets the requirements noted in *Operating System*; however, for security and efficiency reasons, implementers SHOULD NOT use full general purpose GNU/Linux operating system distributions. Implementers SHOULD use distributions specifically built for embedded environments and should excise any unnecessary system components and/or services.

GEISA LEE elements are shown in blue in Figure 9.1 above.

VEE GEISA applications are provided as loadable VEE archives, as detailed in *Virtual Execution Environment*. A VEE conformant GEISA implementation is able to launch the archive in a VEE as necessary and grant the application the rights which the system operator permitted to the application in the application deployment manifest.

The VEE used by the platform implementation is out of scope of this specification, but conformant implementations MUST be able to enforce the permissions and controls the GEISA specification requires, and support the APIs required by the GEISA specification.

Note that GEISA VEE elements are not currently shown in Figure 9.1 above.

General operating system functions such as file system access, math libraries, and network socket APIs are provided by the GEISA LEE or VEE discussed above. The GEISA API facilitates access to platform capabilities that are not serviced by the LEE's or VEE's respective APIs. The GEISA API is described in detail in *Application Programming Interface (API)*. The GEISA API will provide access to metrological data, sensor data, billing data, actuators, and, for GEISA ADM conformant implementations, message exchange with the EMS.

GEISA API conformant platform implementations MUST provide an implementation of the message bus described in *Application Programming Interface (API)* and MUST respond appropriately to all

required API transactions. Connecting the GEISA API implementation to the underlying platform is out of scope for the GEISA specification.

Platform implementers may use any supporting implementation they see fit; however, implementers SHOULD consider the security implications of the implementation and ensure they are robust against both unintended abuse and deliberate attacks.

GEISA API elements are shown in green in Figure 9.1 above.

9.2 Application Isolation

Note

Application Isolation is required of all GEISA execution environments. This section discusses what execution environments must achieve. Requirements specific to a given EE are covered in the respective EE chapters.

GEISA applications shall be isolated from each other and the core platform for the following reasons:

- To ensure that one application cannot impact another application.
- To ensure that one application cannot see the artifacts, resources, data, state, or existence of another application.
- To ensure that applications cannot impact the platform or workloads outside of the GEISA implementation.

Because the GEISA specification defines multiple execution environments, the Application Isolation Implementation (AII) is expected to vary between execution environments and may vary from one platform to another within an execution environment. Regardless of AII (e.g. LXC container, systemd, VEE, etc.), an **authenticated application manifest** shall control access to platform resources and the GEISA API. Additionally, it shall enforce the principle of least-privilege.

The application isolation implementation of GEISA conformant platforms shall ensure that:

- Applications run in independent processes
- Applications run with least privilege
- Application access permissions are deny by default
- Application-to-application communication are denied by default
- Applications cannot access other application's memory or other resources
- Applications do not know about other applications unless explicitly informed

- Applications cannot access the platform's local file-system
- Applications cannot impact the performance of the platform
- Applications cannot impact the stability of the platform
- Applications cannot impact the performance of other applications
- Applications cannot impact the stability of other applications
- Applications cannot create denial-of-service situations
- Resources are fairly distributed when oversubscribed

9.2.1 Application Manifest

The same application manifest is used across all AII. Because of this, all AII SHALL support the same controls, in support of the settings defined in the application manifest, even if their underlying implementation is substantially different. Application manifests are documented in *Application Manifests*.

9.2.2 Networking Control

The AII must control access to the network interface. By default, applications are not granted any network access. The application manifest may indicate that a given application is allowed to natively access the local network interface.

The AII shall control:

- Whether direct network access is allowed.
- Which network interfaces an application may access (none by default), including HAN/LAN and FAN/WAN interfaces.
- The allowed instantaneous bandwidth an application may use.
- The allowed average network volume an application may use over a daily period (e.g., 24 hours).
- Allowed destination peers (IP/proto/port).

9.2.3 API Control

These permissions relate to controlling application access to the GEISA API provided by the platform. See *Application Programming Interface (API)* for details. Each defined API shall have its own set of permissions.

9.2.4 Container Resource Management

Container resource limits shall include the following:

- CPU limit (% of CPU)
- Memory Limit (in 1KiB units)
- Persistent Storage Limit (in 1KiB units)
- Non-Persistent Storage Limit (in 1KiB units)

Note

An Application or Deployment manifest can be modified and re-deployed during an Application upgrade or administrative change. Changes to the resource requirements SHOULD not require an Application restart, but an EE implementation MAY stop, modify, and restart an Application if necessary.

If the persistent storage limit is increased, an implementation MUST honor that change and provide the application with a larger volume or limit without loss of existing persistent data. If that limit is reduced, an implementation SHOULD attempt to honor that change and reduce the volume or limit, however if the Application is using more than the new limit an alarm or exception SHOULD be raised to the EMS and the action MUST be aborted, leaving the Application running with the previous manifest.

10 Conformance

The Grid Edge Interoperability & Security Alliance was established to advance interoperability within the utility industry. Interoperability exists in many forms.

To enable platform vendors, application developers, management system providers, and utilities to validate whether a given system component conforms with this specification, Grid Edge Interoperability & Security Alliance provides an open source conformance test framework.

This framework is licensed under the Apache License and is available in the Grid Edge Interoperability & Security Alliance github repository.

11 Hardware Expectations

The GEISA specification is written to align with the hardware already in use by platform vendors serving electric utilities. This version of the specification refers to hardware *expectations*, rather than hardware *requirements*, because it does not require binary compatibility.[1] Neverthless, it is important to define the expected hardware deployment platform, as it informs a variety of design decisions.

A full GEISA conformant platform, including ADM, API, and LEE or VEE interoperability conformance, should be realizable on:

- ARMv7 CPU with NEON extensions
- 512 MB of RAM
- 1 GB of Flash
- One (or more) network interfaces
- One (or more) metrology interfaces

Note

For the LEE, as this version of the specification does *not* require binary compatibility, but is limited to source-code compatibility only for LEE GEISA applications, platform implementers MAY choose to use alternate CPUs and/or fewer resources for the LEE.

Platform implementers should be aware that:

- It MAY NOT be possible to implement a fully conformant (ADM, API, and EE) platform on a device with fewer resources

[1] Future versions of the GEISA specification may require binary compatibility, though it is likely this will only be done in conjunction with a GEISA community reference implementation.

- If GEISA is extended to include binary compatibility in the future, alternate architectures will likely create additional management work for end users (VEE GEISA applications are agnostic to underlying CPU architecture, but are source-code compatible only in GEISA 1.0.)

The vision is that the GEISA EE runs on a wide range of hardware platforms with various capabilities:

- Device types such as smart meters, load switches, EV chargers, etc.
- Single and multi-core CPUs
- Support for processor extensions and coprocessors such as GPUs
- RAM sizes greater than the minimum 512MB target
- Storage sizes greater than the minimum 1 GB target
- Multiple networking interfaces such as mesh, Wi-Fi, cellular, etc.
- Metrology interfaces to provide voltage, current, etc.
- Actuators such as relays and contactors
- Sensors including temperature, humidity, location, vibration and more
- Hardware watchdogs
- Generic hardware interfaces such as GPIO, ADC, SPI, etc.

Platform providers offering a GEISA EE MUST provide a toolchain which allows application developers to build applications written for the GEISA EE.[2]

11.1 Metrology

GEISA specifically addresses edge computing enviroments used by electric utilities. It is assumed there is a source of metrological data available on GEISA devices. The GEISA API provides a mechanism for GEISA applications to obtain details regarding the metrological capabilities of the device (see *Platform Discovery*)

Note

The GEISA API is not expected to provide metrological information when used on a non-metrological device; however, some platforms without local metrology MAY support providing remote metrological data through their local GEISA API

[2] GEISA LEE specifically assumes a GNU/Linux environment (see *Operating System*), such that it is likely that the tool chain in use is open-source. If a vendor is using a commercial tool chain, there is no requirement that the platform vendor provide a license; however, they MUST provide information regarding where third parties can purchase the necessary tool chain and the version of the tool chain in use, and they must provide any supporting files required to allow the tool chain to be used to compile applications for their platform.

Metrological hardware, at a minimum, should be able to provide:[3]

- Instantaneous RMS Voltage Reading
- Instantaneous RMS Current Reading
- Frequency

When coupled with a billing register, metrological hardware may also be able to provide derived quantities such as demand values, cummulative values, and interval values.

Metrological hardware may also be able to provide waveform data. See *Waveform Data* for additional details.

When waveform data is provided, at a mininum, it SHALL provide:

- 128 samples per cycle (7.68 kHz at 60 Hz or 6.4 kHz at 50 Hz AC frequencies)
- 16 bits per sample

Platforms may provide higher sampling rates (e.g. 128, 256, 512, 16,384 samples per cycle or more) and greater sampling resolution (e.g. 32 bits per sample).

Depending on the device, waveform data may be provided for voltage, current, or both. Waveform data may be split-phase, phase-to-phase, or phase-to-neutral. The GEISA API allows applications to obtain details about the waveform data available on the platform so that they can correctly interpret the information the platform exposes through the API.

11.2 Sensors

A GEISA platform MAY optionally provide access to one or more sensors. Sensors are exposed through the GEISA API  .

Examples of sensors that may be provided include:

- Temperature Sensor
- Humidity Sensor
- Switch Sensor
- GNSS Location
- Vibration / accelerometer

[3] Note: inductively powered devices, such a remote fault indicators, may lack a ground reference and thus may be unable to provide voltage data. Similarly, some devices may lack a current sensor and may thus be limited to voltage only. Electric meters are expected to be able to provide both.

11.3 Actuators

A GEISA platform MAY optionally provide one or more actuators. Actuators are exposed through the GEISA API.

Examples of actuators that may be provided include:

- Service disconnect switch
- DER disconnect switch
- Demand response relay

12 Application & Device Management

GEISA provides a uniform mechanism for managing edge devices and the applications that run on top of them. ADM conformant devices from different platform vendors can be managed by a single ADM conformant edge management system. See *System Architecture* for more discussion of the general concept.

ADM conformance devices support a set of standard transactions using the Open Mobile Alliance Lightweight M2M [LWM2M] transaction definitions, over the IETF CoAP [RFC7252] protocol. ADM conformant platform implementations MUST support the LWM2M transactions detailed in this chapter and the supporting device behaviors described. ADM conformant edge management systems MUST support the LWM2M transactions detailed in this chapter and the supporting edge management system behaviors described.

Note

ADM conformance does not preclude platforms or edge management systems from supporting additional or supplementary management protocols.

Note

Platform vendors may offer an ADM conformant device without offering an ADM conformant edge management system or an ADM conformant edge management system without offering an ADM conformant device.

As described in the GEISA *Application Programming Interface (API)*, GEISA provides a default

Application Messaging and Configuration mechanism. ADM conformant platforms transport these messages over CoAP. ADM conformant edge management systems expose these messages over a local ReST API, allowing other systems to easily consume messages sent by edge applications or to send messages to edge applications.

12.1 OMA Lightweight M2M

GEISA Application & Device Management uses the Open Mobile Alliance Lightweight M2M [LWM2M] protocol for application management. Specifically,

- ADM conformant platforms and Edge Management Systems (EMS) SHALL support LWM2M v1.1 or greater. [LWM2M-Core]
- ADM conformant platforms and EMS MAY support LWM2M versions greater than v1.1; however, they MUST retain compatibility with v1.1.
- ADM conformant platforms and EMS SHALL support CoAP over UDP for message transport. [LWM2M-Transport]
- ADM conformant platforms and EMS SHALL support DTLS for CoAp security.
- ADM conformant platforms and EMS SHALL support the Bootstrap, Registration, Device Management and Information Reporting interfaces.
- ADM conformant platforms and EMS SHALL support the Mandatory resources of the most recent version of the required LWM2M Objects.
- ADM conformant EMS shall support the following LWM2M Objects
  - ID 0 – Security
  - ID 1 – Server
  - ID 3 – Device
  - ID 4 – Connectivity Monitoring
  - ID 5 – Firmware Update
  - ID 6 – Location
  - ID 9 – Software Management
  - ID 10 – Cellular Network Connectivity
  - ID 11 – APN Connection Profile
  - ID 12 – WLAN Connectivity
  - ID 13 – Bearer Selection
  - ID 20 – Event Log
  - ID 504 – Remote SIM Provisioning

  - ID 3600 – GEISA App Messaging
  - ID 3601 – GEISA Host Monitoring
  - ID 3602 – GEISA App Accounting
  - ID 3603 – GEISA Wi-SUN Radio Management
- ADM conformant platforms shall support the following LWM2M Objects
  - ID 0 – Security
  - ID 1 – Server
  - ID 3 – Device
  - ID 4 – Connectivity Monitoring
  - ID 5 – Firmware Update
  - ID 9 – Software Management
  - ID 20 – Event Log
  - ID 3600 – GEISA App Messaging
  - ID 3601 – GEISA Host Monitoring
  - ID 3602 – GEISA App Accounting
- ADM conformant platforms with 3GPP network interfaces SHALL support the following LWM2M Objects:
  - ID 10 – Cellular Network Connectivity
  - ID 11 – APN Connection Profile
  - ID 12 – WLAN Connectivity
  - ID 13 – Bearer Selection
  - ID 504 – Remote SIM Provisioning
- ADM conformant platforms with GNSS interfaces SHALL support the following LWM2M Objects:
  - ID 6 – Location
- ADM conformant platforms with Wi-SUN interfaces SHALL support the following LWM2M Objects:
  - ID 3603 – Wi-SUN Radio Management

12.2 Bootstrapping

As discussed in the LWM2M core specification [LWM2M-Core] section 6.1, there are four ways to bootstrap a device so that it knows how to contact its LWM2M server for management:

- Factory Bootstrap
- Bootstrap from Smartcard
- Client Initiated
- Server Initiated

Using Factory Bootstrap, devices are provisioned during manufacturing with security credentials and Management Server (edge management system) information. Upon initial field installation, devices will automatically connect directly to their Management Server (Edge Management System/EMS) to perform registration.

Under a Client-Initiated Bootstrap, shown below in Figure 12.1, devices are provisioned during manufacturing with Bootstrap Server (BS) information. Upon field installation, devices initiate the connection to the Bootstrap Server for the provisioning of Security credentials (object 0) and Management Server information (and subsequent registration).

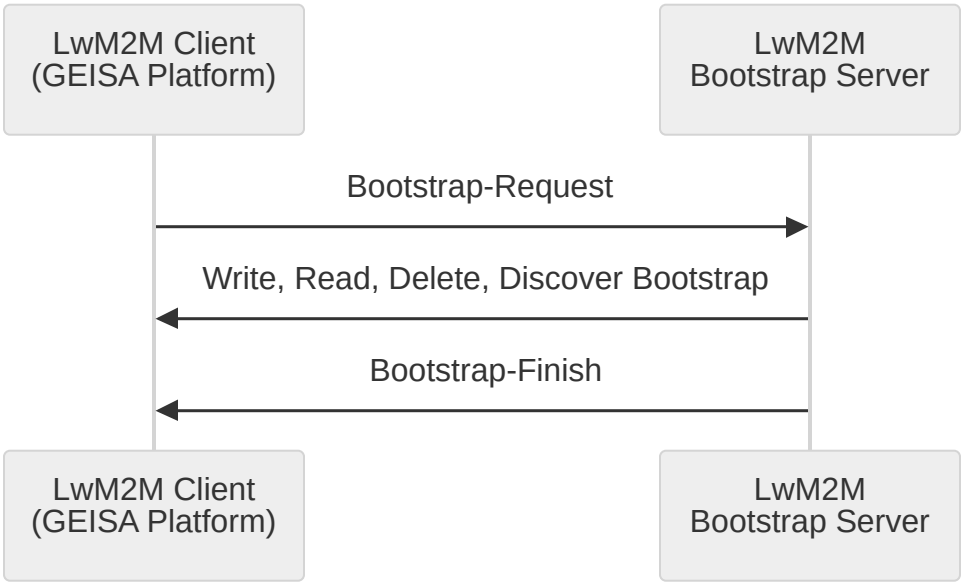


Figure 12.1: Client Bootstrap

As noted in [LWM2M-Core], GEISA ADM conformant devices SHALL support Client Initiated bootstrapping. GESIA ADM conformant devices MAY support other bootstrapping methods. For mass deployed devices, like smart meters, Factory Bootstrapping is usually preferred by system operators.

GEISA ADM conformant devices SHALL provide an accessible out-of-band mechanism which allows the system operator to preprovision the URL of the CoAp Bootstrap-Server, the Bootstrap-Server Account credentials, and any other data needed to complete the boostrapping process (e.g. certificates, keys, etc.).

Note

GEISA ADM Conformant EMS are not required to provide a Bootstrap Server. If a platform provider ships devices set to use client intiated bootstrap, it is expected that they will inform the purchaser in advance or provide a bootstrap server.

Note

ToDo: LWM2M Bootstrapping can include an endpoint client name. This is optional if the identifier provided in the security protocol is sufficient. We need to agree on the security mechanism and discuss whether the identifiers it provides are sufficient.

12.3 Registration

GEISA ADM conformant devices SHALL attempt to register with a GEISA conformant edge management system on startup. Registration uses the LWM2M *Client Registration Interface*.

The LwM2M Registration process allows a GEISA ADM conformant platform to inform the EMS of the device's configuration and capabilities, and to request management by the EMS. It also informs the EMS of the current network address of the device, which is likely dynamically assigned by the network. During Registration, the LwM2M Client in the platform reports to the Edge Management System the list of LwM2M Objects supported by the EE and its currently instantiated Object Instances.

GEISA ADM conformant devices SHALL use the Registration Update, a lightweight empty Registration packet sent to the EMS, for the following:

- A periodic heartbeat from Client to Server to maintain the existing management session.
- If any previous Registration information changes, for example:
  - GEISA platform IP Address changes
  - The objects supported by the platform changes after a firmware upgrade

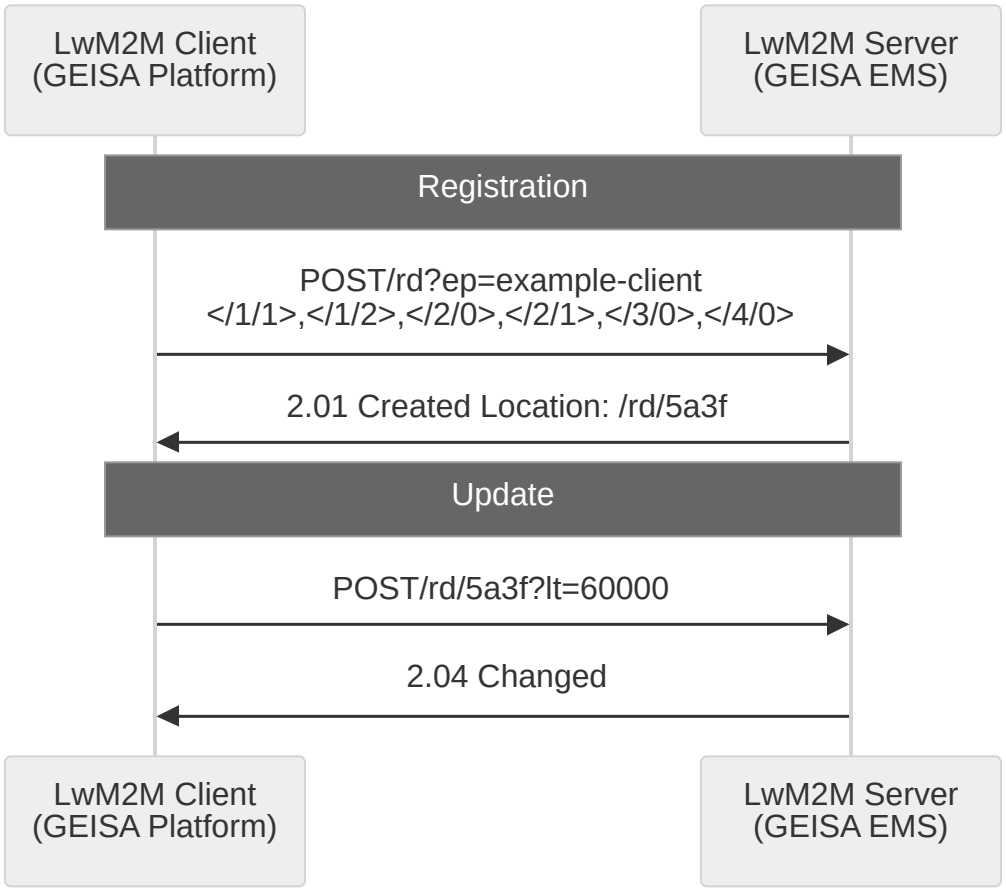


Figure 12.2: LWM2M Registration and Registration Update

12.4 Application Manifests

GEISA's application management system follows a model similar to Amazon IoT Greengrass or Microsoft Azure IoT in that applications have a recipe, manifest, or other set of metadata describing the requirements and dependencies.

GEISA handles application meta-data by defining two manifests for each application: a vendor manifest and an operator manifest.

The vendor application manifest provides information about the application to the operator and the edge management system, including any external dependencies and minimum resource requirements. The vendor manifest is used by a GEISA ADM conformant edge management system to inform the operator about the application and to facilitate the secure import of the application.

The operator application manifest is based on the vendor application manifest allowing the operator to customize and tune the application permissions and resource allocations to match their execution

environment, intended use case, and security requirements prior to deployment to the execution environment.

GEISA vendor and operator application manifests SHALL include:

- Vendor assigned Application ID
  - Vendor assigned application IDs use *https://en.wikipedia.org/wiki/Reverse_domain_name_notation Reverse DNS Name Notation*, similar to Java packages
  - GEISA recommends `tld.companyname.geisa.appname` for application IDs, for example, `org.lfenergy.geisa.waveformanalyzer`.
- Name of the application
- Description of the application
- Version Number of the application
- Hash of the application image
  - The GEISA EE shall not activate an application unless the hash of the image matches the hash in the meta-data

GEISA vendor manifests SHALL include:

- Compatibility:
  - GEISA API Minimum Version
  - GEISA LEE Minimum Version (null for unsupported)
  - GEISA VEE Minimum Version (null for unsupported)
  - Waveform Access Required (boolean)
  - GEISA LEE CPU arch string as returned by the Linux arch command:
    - ARM 32-bit: armv7l, armv6l
    - ARM 64-bit: aarch64
    - RISC-V 32-bit: riscv32
    - RISC-V 64-bit: riscv64
    - x86 32-bit: i686
    - X86 64-bit: x86_64, amd64
  - GEISA LEE C library required:
    - glibc
    - musl
    - uClibc

        * uClibc-ng
    - For VEE, a string of the JVM version:
        * Java 8: java8
        * Java 11: java11
        * Java 17: java17
        * Java 21: java21
- System Resources Required:
    - CPU usage in percent
    - RAM in KiB
    - Persistent Storage in KiB
    - Non-persistent Storage in KiB
    - Max threads/processes
- Off-Device Communication
    - Each application that performs off-device communication MUST specify the classes, destinations, and daily volume of that communication.
    - See *Off-Device Communication* for details.
    - Daily volumes are specified in Byte units, and daily messages are in message units.

Note

Application messaging is provided through the API and does not require direct access to a network interface.

Note

Requests for IP socket communication in a vendor manifest are subject to operator approval in the deployment manifest. Operators may approve, narrow, replace, or deny requested destination classes, endpoints, ports, and volume limits according to deployment policy.

- External Dependencies
    - GEISA applications should be as self-contained as possible, with all necessary dependencies, except for the base libraries provided by the EE, contained with the application artifacts.

  - GEISA does not provide a mechanism for loading arbitrary packages. The external dependencies element in the manifest is used exclusively to indicate that one application depends on another.
  - Vendor-assigned Application ID of the application this application depends on.

- Default Application Configuration
  - GEISA applications may need basic information to initialize such as the URL of a server, or settngs such as the frequency of reportng. The default application configuration provides an initial set of values that can be used by the system operator when creating the operator manifest.

Note

The system operator should be able to change the configuration information without needing to redeploy the application.

- Default Launch Strategy
  - Includes details such as whether the application should automatically be restarted if it fails, and how many failures with a given period of time constitues a permanent failure.
    * Auto restart (boolean)
    * Max failures – number of failures within the restart period after which the application will not be restarted
    * Restart period – elasped time, in minutes, before the failure count is reset. The first failure starts the timer. If the configured maximum number of failures occurs before the restart period is over, the application is not restarted. Otherwise, the failure count is reset.
- Vendor Signature
  - Base64 Encoded Signature of the compact JSON encoding of the vendor application manifest.

Note

ToDo: Add details on the signature mechanism.

Here is an example of an vendor application manifest.

```
{
  "geisa-vendor-app-manifest": {
    "org.lfenergy.geisa.HelloWorld": {
      "author": "Some Company",
```

(continues on next page)

(continued from previous page)

```
    "name": "Hello World Application",
    "description": "Killer application that writes 'hello world' to the log",
    "version": "1.0.0",
    "artifacts": {
      "image-size": 748340,
      "uncompressed-size": 2494464,
      "image": "helloworld-1.tgz",
      "sha256": "00beeaeeca59f9177d88a13132f7c0686616fe728d85f20ddbd15352abd10988"
    },
    "compatibility": {
      "GEISA-API": "1.0.0",
      "GEISA-LEE": "1.0.0",
      "GEISA-VEE": null,
      "CPU": "aarch64"
      "LIB": "musl"
    },
    "resources": {
      "app-cpu": 30,
      "app-ram": 40,
      "storage-persist": 20,
      "storage-nonpersist": 5,
      "threads": 50,
      "AMI": false,
      "HAN": true,
      "waveform": true
    },
    "communication": {
      "message": {
        "daily-messages": 30
      },
      "operator": {
        "daily-volume": 2048,
        "outbound": [
          "tcp:[3fff:421:32::/48]:443",
          "udp:[3fff:421:2:661::/64]:4242",
          "tcp:198.51.100.0/24:999"
        ]
      },
      "internet": {
        "daily-volume": 51200,
        "outbound": [
```

(continues on next page)

(continued from previous page)

```
            "tcp:[2001:db8:44:12::/64]:443",
            "udp:203.0.113.66:2256"
          ]
        }
        "local": {
          "outbound": [
            "tcp::9999",
            "tcp::502"
            "udp::51234"
            "udp::5540"
          ]
          "inbound": [
            "tcp::5540",
            "udp::5540",
          ]
          "inbound-multicast": [
            "255.255.255.255",
            "224.0.0.251",
            "ff02::fa",
            "ff02::fb",
          ]
        }
      },
      "external-dependencies": [
        null
      ],
      "default-configuration": {
        "knob": 36,
        "setting": "blue",
        "turbo encabulator active": true
      },
      "default-launch-strategy": {
        "auto-restart": true,
        "max restarts": 5,
        "restart period": 60
      }
    }
  }
}
```

12.5 Device Management

Device management within GEISA allows system operators to track, update, restart, and reset edge environments. As discussed under *Registration*, when a GEISA ADM conformant device starts, it registers with the GEISA Edge Management System (EMS).

The Registration context allows the EMS to track the general status of the edge device fleet. To facilitate effective management, during Registration an ADM conformant EMS SHALL Read or Observe as appropriate all required GEISA objects advertised by the device platform, including:

Object ID	Object Name	Information
3	Device	Mfg, Model, S/N, Firmware Version, System Clock, Storage, etc.
4	Connectivity Monitoring	IP Address, Link Quality, LwM2M Network Bearer, etc
6	Location	GNSS location
10	Cellular Connectivity	3GPP connection management
11	APN Connection Profile	APN connection management
12	WLAN Connectivity	Wi-Fi Radio interface management
13	Bearer Selection	LwM2M bearer selection management
20	Event Log	System Log and App-specific Log retrieval
504	Remote SIM Provisioning	eSIM profile management: reporting, swap, add/delete
3600	GEISA App Messaging	App data reporting on the uplink, App config on the downlink
3601	GEISA Host Monitoring	CPU, RAM, process, context switch, file handle observability
3602	GEISA App Accounting	System-level and App-level bandwidth usage and optional throttling

To avoid the overhead of full re-Registration during normal session continuance, ADM conformant devices SHALL send a lightweight Registration Update prior to the expiration their Registration Lifetime in order to maintain their Registration context with the server. Although not directly specified in the LwM2M protocol, ADM conformant devices SHOULD send a Registration Update after expiration of 50% of the Registration Lifetime, similar to the timing strategies of RFC 2131. Upon receipt of a Registration Update, an ADM conformant EMS SHALL restart the Lifetime expiration timer for the device. ADM conformant devices SHALL only perform a full re-Registration under the following conditions:

- Registration Lifetime expired
- Client or Server loses the Registration state
- Change of Server URI or Security Context

- Re-Bootstrap
- Client reachability change (IP Address, NAT binding, Endpoint name, etc.)

ADM conformant devices that maintain their Registration state across reboots are not required to perform a full re-Registration after a reboot or power restoration.

Device management is also used to perform platform-level firmware updates. Firmware updates are performed using LWM2M Object 5, Firmware Update.

ADM conformant GEISA devices shall support device reboots as well as device factory resets, using LWM2M Object 3. Factory resets of an ADM conformant device shall remove all installed applications and any associated application data. During factory reset, the EMS MAY specify management of local LDevID credentials by submitting an argument with the Execute /3/0/5 operation:

- No Argument or Argument = 0 indicates that the Client MUST preserve its IDevID upon factory reset.
- Argument = 1 indicates that the Client MUST preserve both IDevID and LDevID(s) upon factory reset.

The LwM2M Device Management and Service Enablement interface exposes the facility to perform device, application, and network management operations on an ADM conformant GEISA platform:

- **Discover** – Used by a LwM2M Management Server to retrieve the list of Resources instantiated in each Object instance. Data (Resource Values) is not returned.
- **Read** – Used by an EMS to retrieve Resource data values (e.g., sensor reading). Reading may be performed at various levels: Resource Instance, entire Resource, Object Instance, entire Object
- **Read-Composite** – Used by an EMS to retrieve multiple Resources/Objects in single CoAP request.
- **Write** – Used by an EMS to modify Device configuration.
  - CoAP PUT is used to Replace the Object Instance or Resource(s) with the new values provided.
  - CoAP POST is used for Partial Update to update the Resources with the new values provided, leaving other existing Resources unchanged.
- **Write-Composite** - Used by an EMS to update multiple Resources/Objects in single CoAP request.
- **Execute** – Used by an EMS to invoke commands on the platform (e.g., Factory Reset, Activate Edge App).
- **Create** – Used by an EMS to create new Object Instances on the LwM2M Client of the platform.
- **Delete** – Used by an EMS to delete Object Instances on the LwM2M Client of the platform.

- **Write-Attributes** – Used by an EMS to set Notification triggers for an Observe of a Resource/Object (e.g., Only send a Notification every two hours, only send a Notification if the observed value has changed by more than X).

These operations are performed using the following CoAp methods:

Operation	CoAp Method	Path	Success	Failure
Read	GET Accept: Content Format ID	/{Object ID}/{Object Instance ID}/{Resource ID}	2.05 Content	4.00 Bad Request, 4.01 Unauthorized, 4.04 Not Found, 4.05 Method Not Allowed, 4.06 Not Acceptable
Discover	GET Accept: applicatio format	/{Object ID}/{Object Instance ID}/{Resource ID}	2.05 Content	4.00 Bad Request, 4.01 Unauthorized, 4.04 Not Found, 4.05 Method Not Allowed,
Write	PUT Content Format:	/{Object ID}/{Object Instance ID}/{Resource ID}	2.04 Change 2.31 Continue	4.00 Bad Request, 4.01 Unauthorized, 4.04 Not Found, 4.05 Method Not Allowed, 4.06 Not Acceptable 4.08 Request Entity Incomplete 4.13 Request Entity Too Large
Write	POST Content Format:	/{Object ID}/{Object Instance ID}	2.04 Change 2.31 Continue	4.00 Bad Request, 4.01 Unauthorized, 4.04 Not Found, 4.05 Method Not Allowed, 4.06 Not Acceptable 4.08 Request Entity Incomplete 4.13 Request Entity Too Large
Write Attri	PUT	/{Object ID}/{Object Instance ID}/{Resource ID} ?pmin={minimum period}&pmax={maximum period} &gt={greater than}&lt={less than}&st={step}	2.04 Change	4.00 Bad Request, 4.01 Unauthorized, 4.04 Not Found, 4.05 Method Not Allowed,
Execute	POST	/{Object ID}/{Object Instance ID}/{Resource ID}	2.04 Change	4.00 Bad Request, 4.01 Unauthorized, 4.04 Not Found, 4.05 Method Not Allowed,
Create	POST Content Format:	/{Object ID}	2.01 Created	4.00 Bad Request, 4.01 Unauthorized, 4.04 Not Found, 4.05 Method Not Allowed, 4.06 Not Acceptable
Dele	DELETE	/{Object ID}/{Object Instance ID}	2.02 Deleted	4.00 Bad Request, 4.01 Unauthorized, 4.04 Not Found, 4.05 Method Not Allowed,

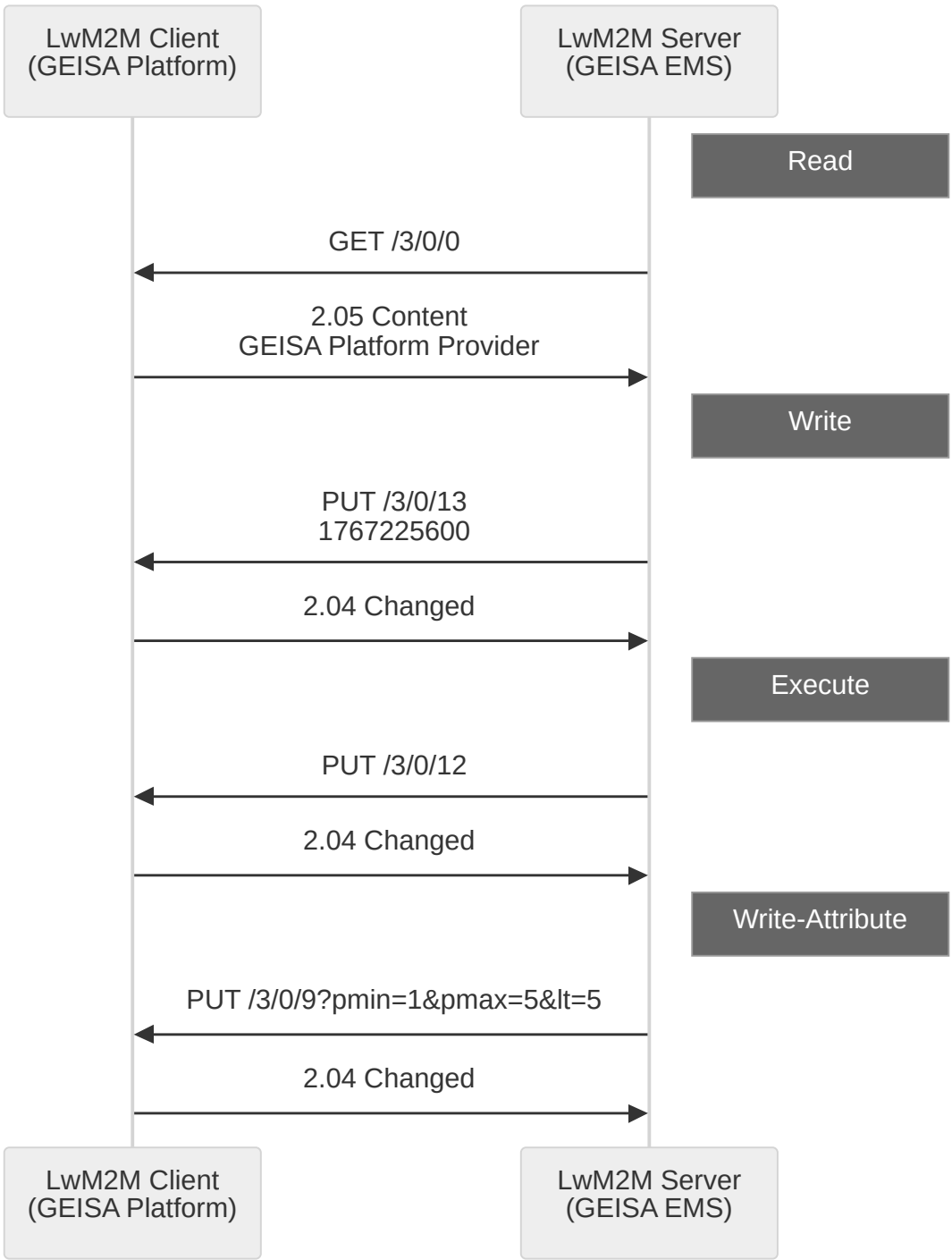


Figure 12.3: Device Management Operations

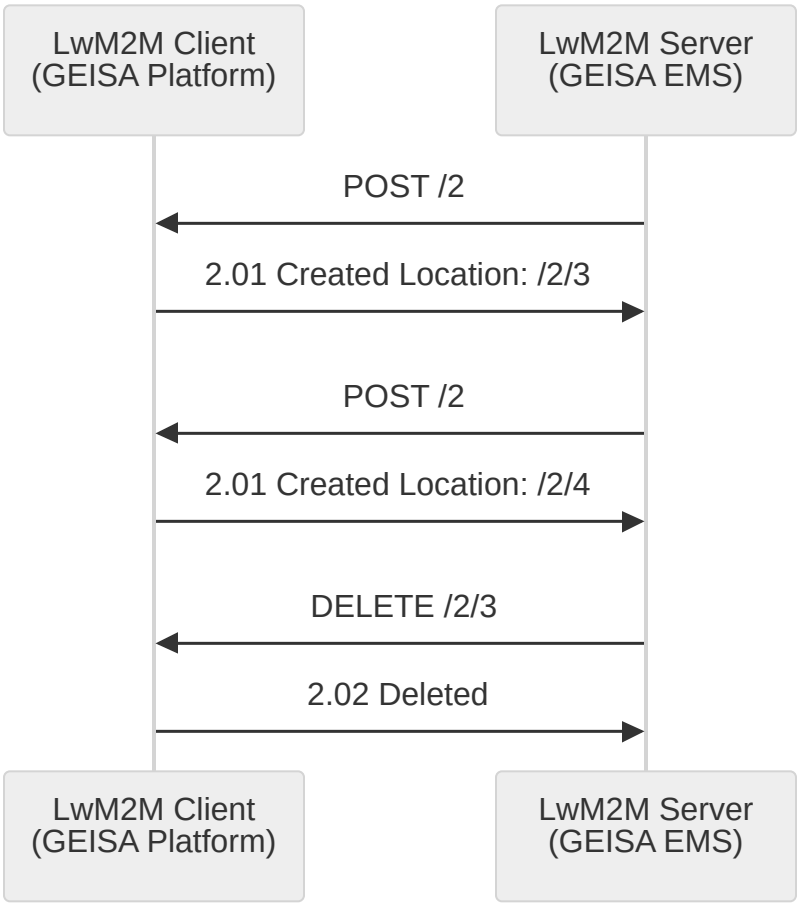


Figure 12.4: Object Creation and Deletion

12.6 Firmware Management

Firmware management is the process of updating the core operating system for an ADM compliant GEISA platform.

The LwM2M singleton object `/5/0 Firmware Update` represents the firmware update *process*, not simply the firmware image stored on the device. Object `/5/0` is advertised during initial Registration, and object `/5/0` is not deleted by the GEISA platform client after an upgrade; only the `/5/0/3 State` and `/5/0/5 Update Result` resources are reset after an upgrade to reflect the new baseline.

The multi-step Firmware Update transaction typically involves the EMS *Observe* `/5/0/3 State` so that the GEISA platform client will provide asynchronous updates of client state changes to the EMS, after which the EMS will proceed with subsequent steps in the transaction. Although it is technically possible to *Observe* resources `/5/0/3` and `/5/0/5` following initiation of the update

transaction, in order to avoid transaction dead locks or inconsistent state due to race conditions, and to provide optional support for resource `/5/0/14 Automatic Upgrade at Download`, a GEISA ADM conformant EMS SHALL perform the following:

- *Observe* `/5/0/3 State` when object `/5/0` is advertised during GEISA platform Registration
- *Read* `/5/0/5 Update` Result after Notification that `/5/0/3 State = Idle`

Depending on EMS capabilities, LwM2M allows either of the following methods for firmware image distribution to the GEISA platform:

- **PUSH** (CoAP block-wise transfer) via the *Write* operation of the opaque binary image to resource `/5/0/0` (Package)
- **PULL** via the *Write* operation to resource `/5/0/1` (Package URI), allowing the GEISA platform to download via CoAP or HTTP as soon as practical.

The following sequence diagrams provide two examples of GEISA conformant upgrades of the platform firmware. The first example demonstrates PUSH of the firmware image to `/5/0/0 Package`, as shown in Figure 12.5 followed by the EMS performing a manual *Execute* `/5/0/2` to trigger the upgrade in Figure 12.6.

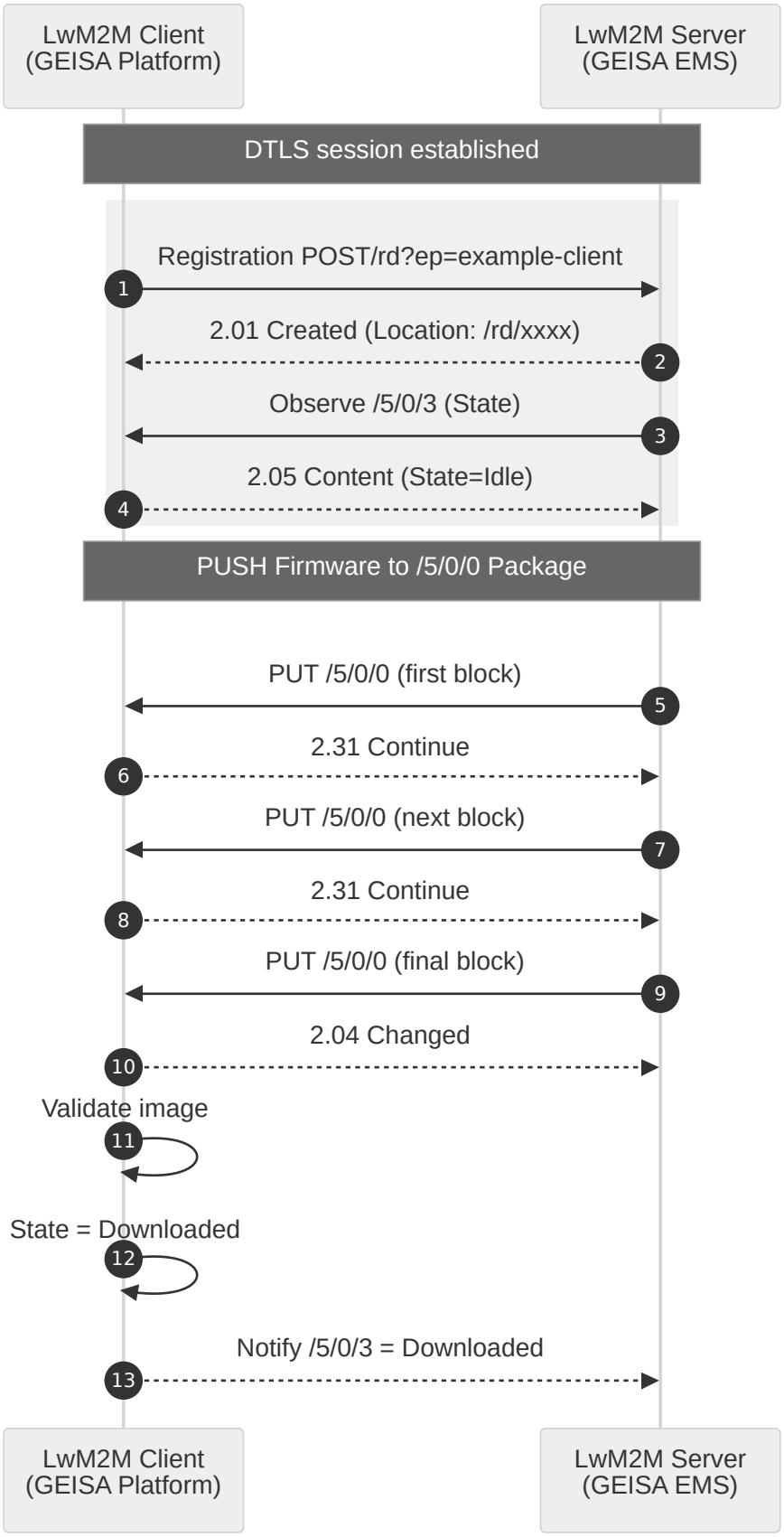


Figure 12.5: Firmware Over-the-Air Push Update

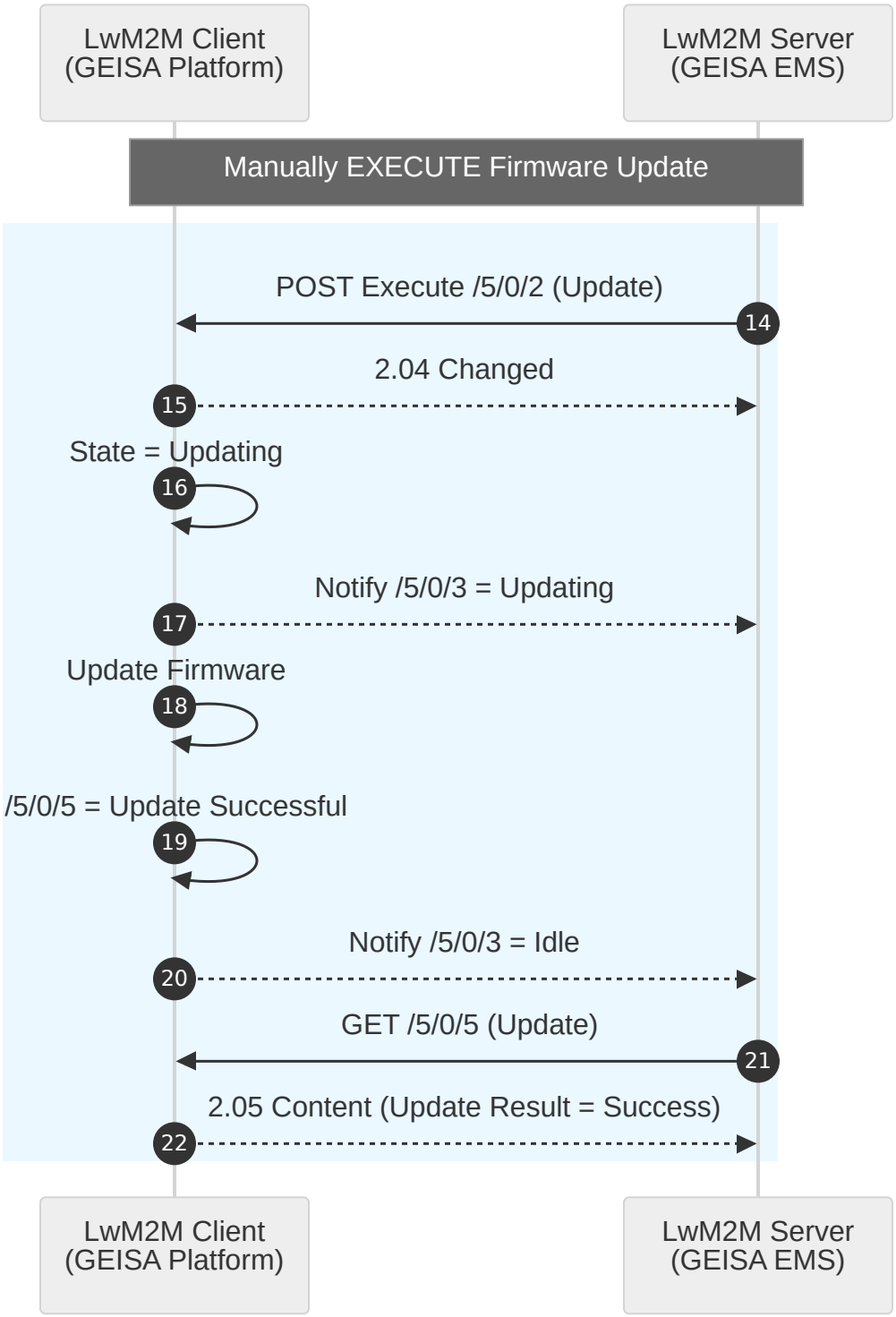


Figure 12.6: Firmware Over-the-Air Manual Activate

The second example, as shown in Figure 12.7, demonstrates the PULL of the firmware image from the URL set by the EMS into `/5/0/1 Package URI`. As shown in Figure 12.8, the ADM conformant

GEISA platform automatically executes the upgrade after the download per the EMS setting `/5/0/14 Automatic Upgrade = True`.

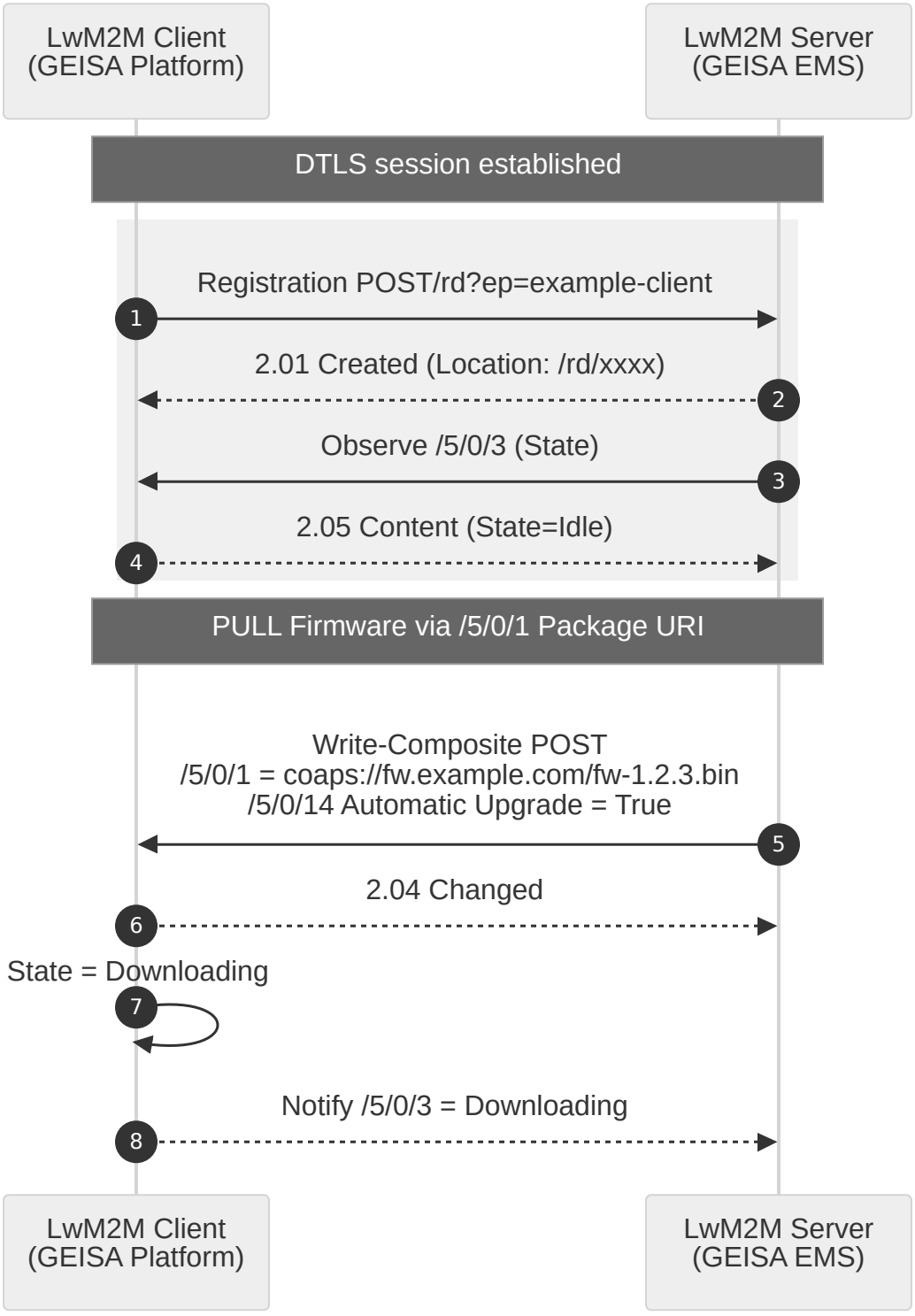


Figure 12.7: Firmware Over-the-Air Pull Image Initiate

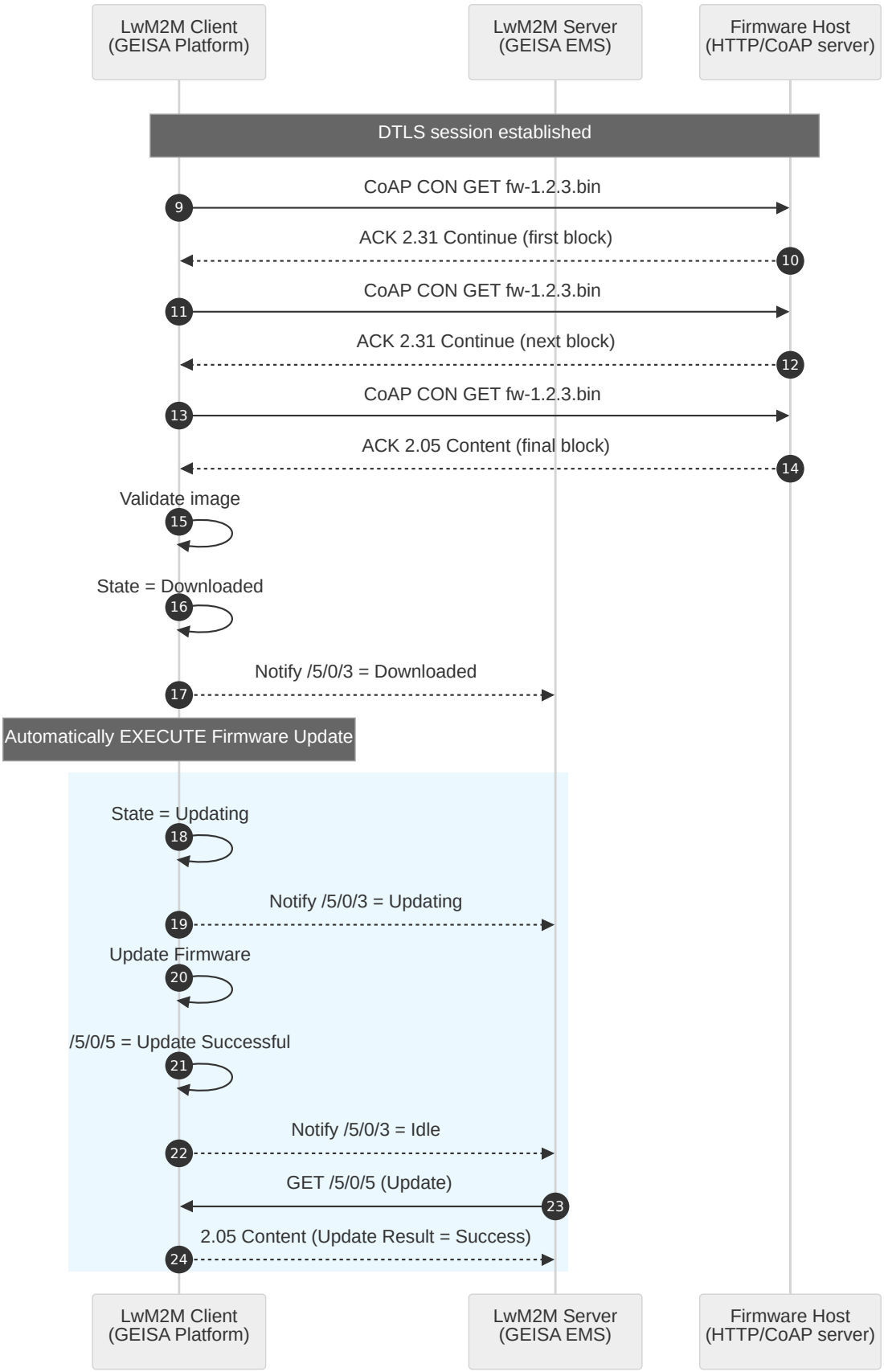


Figure 12.8: Firmware Over-the-Air Pull Automatic Download and Update

12.7 Application Management

Application management is the process of deploying, activating, deactivating, and decommissioning applications on a GEISA ADM compliant platform, for execution within an execution environment.

12.7.1 Software Management

The LwM2M `/9/x Software Management` object SHALL be used to manage the Installation and Activation of containerized edge applications running in the GEISA EE. In contrast to the Firmware Update object, each instance of the multi-instance Software Management object represents a distinct edge application *Package* installed in the EE. The format of the edge application Package is defined in *Base Filesystem* for LEE conformant systems and SHALL be composed of the following components:

- X.509 Public Key Certificate used to verify the digital signature in the Package
- Digital Signature across the Edge Application Manifest and Edge Application Binary
- *Edge Application Manifest*
- Edge Application Binary

To minimize edge app container sizes, applications are encouraged to dynamically link against the libraries provided by the base GEISA environment rather than providing their own. Consideration for the management of the base libraries and/or package dependencies will be deferred to a future release; at this time, no consideration is made for the use of LwM2M object `14 Software Component`.

App Installation and Activation

Similar to *Firmware Update*, the LwM2M spec permits edge app packages can be transferred to the EE via either of the following methods:

- PUSH via *Write* of the opaque package to `/9/x/2 Package`
- **PULL via *Write* to resource `/9/x/3 Package URI` for the GEISA platform**
  to download via CoAP/HTTP as soon as practical

In contrast to Firmware Update, the `Software Management object 9` does not support the concept of automatic Installation or Activation. Both operations of Installation and Activation are manually executed by the EMS, following successful package download/verification and successful package install, respectively.

During creation of an `Software Management object 9` instance to install an edge app, an ADM conformant EMS SHALL use the AppID as the object 9 instance number, in order to simplify subsequent app management operations (start, stop, purge, etc.).

The following example demonstrates GEISA compliant edge app installation and activation:

1. PULL download of the edge app package from the URL set by the EMS into `/9/x/3 Package URI`, shown in Figure 12.9.
2. The EMS performing a manual *Execute* `/9/x/4` to trigger edge app Installation following successful app download and verification, shown in Figure 12.10.
3. The EMS performing a manual *Execute* `/9/x/10` to trigger edge app Activation following successful app installation, shown in Figure 12.10.
4. The EMS performing a manual *Execute* `/9/x/6` to trigger Uninstall of the edge app (not shown).

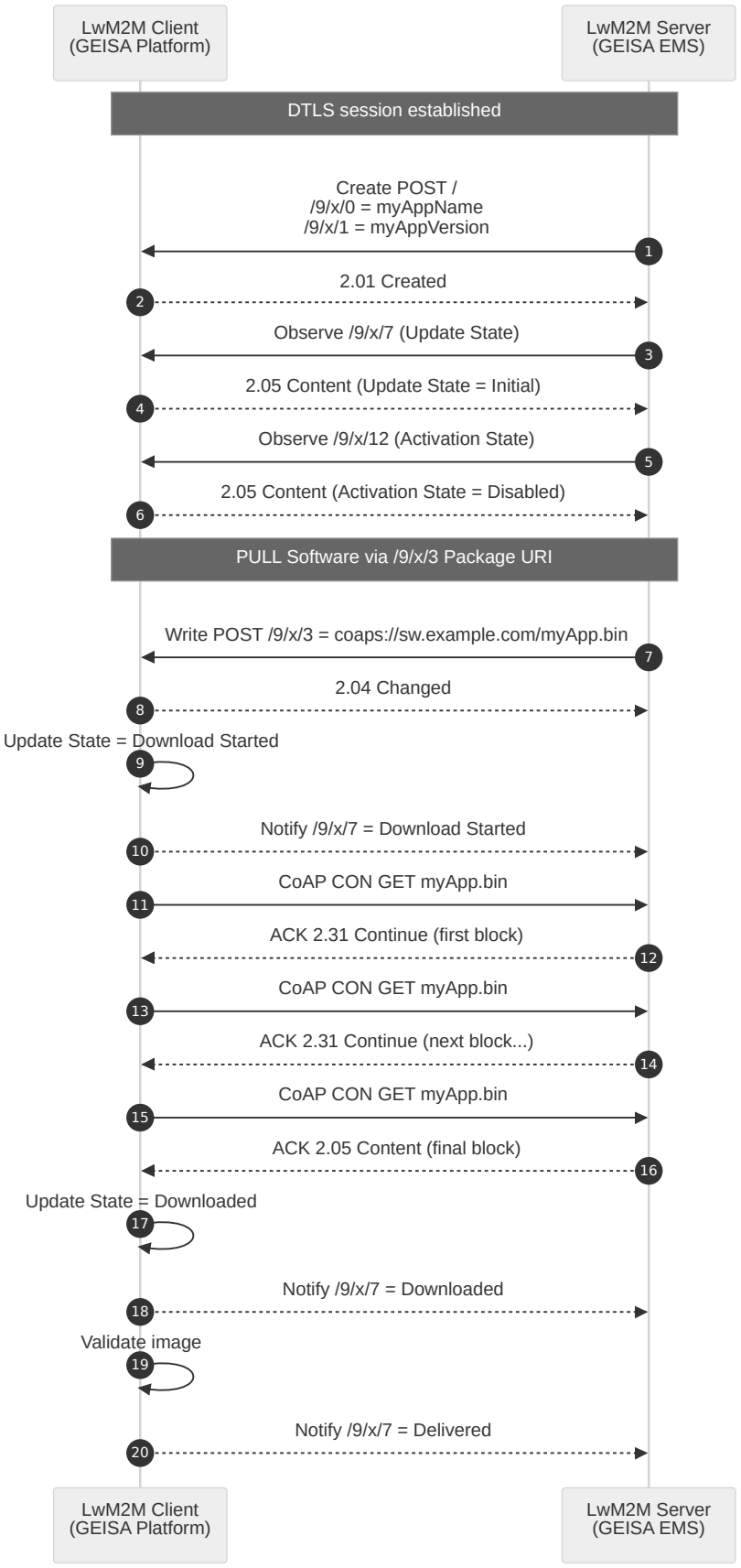


Figure 12.9: Edge App Image Pull

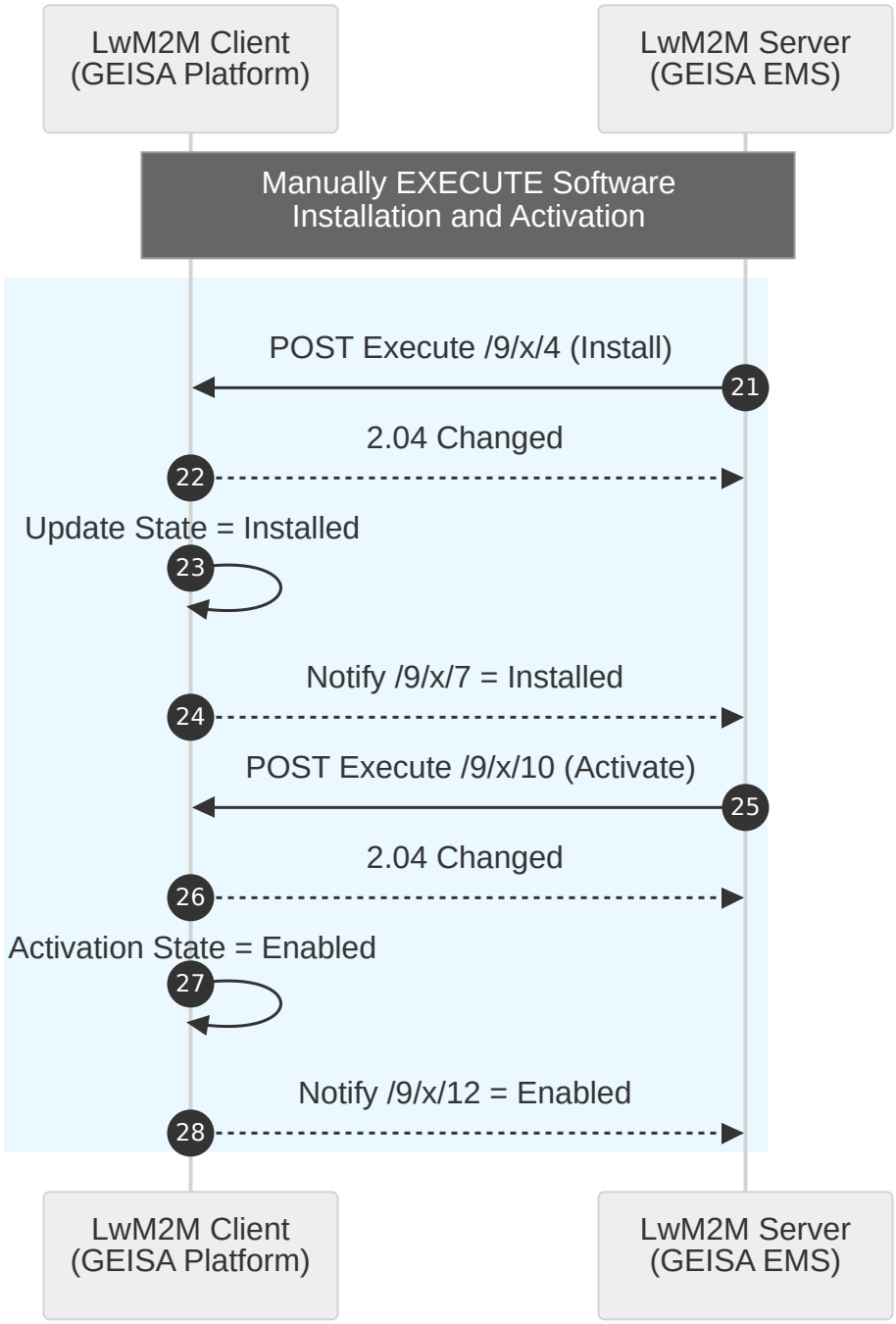


Figure 12.10: Edge App Install and Activate

The Software Management object Activation state machine defines the ability to use an app but does not address app execution state semantics:

- When the current state is set to ACTIVE, the installed software can be used by the LwM2M Client.

- When the current state is set to INACTIVE, the LwM2M Client MUST NOT use the installed software.

App Execution State

Version 1.1 of the Software Management object adds resources for an EMS to control the *Execution State* of an edge application:

Resource ID	Operation	Data Type	Description
19	Execute		Start Application. Only available when Activation State = Enabled.
20	Execute		Stop Application. Only available when Activation State = Enabled.
21	Read	Integer	Execution Status. 0 = Stopped. 1 = Running.

App Purge

Version 1.1 of the Software Management object adds an executable resource for an EMS to remotely purge local data from an instance of an edge application installation:

Resource ID	Operation	Data Type	Description
22	Execute		Purge Data. Deletes existing local application data without modifying the app installation/activation states.

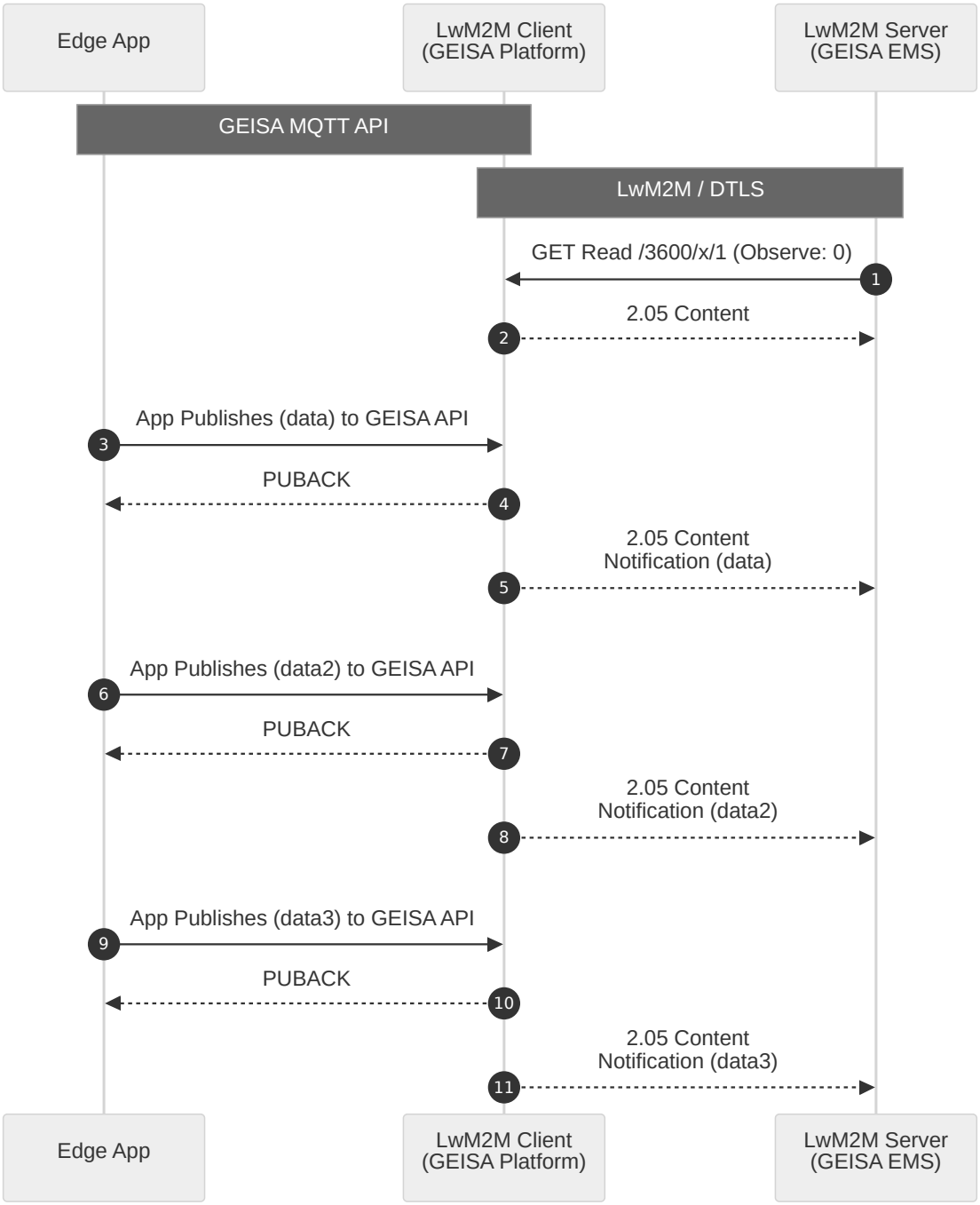
12.8 Application Messaging and Configuration

GEISA LwM2M object 3600 provides a bi-directional messaging facility for applications, with distinct resources defined for Client to Server messaging and Server to Client messaging. Object 3600 SHALL be used by ADM compliant platforms and EMS for sensor data reporting on the uplink (Client to Server) and edge app configuration on the downlink (Server to Client).

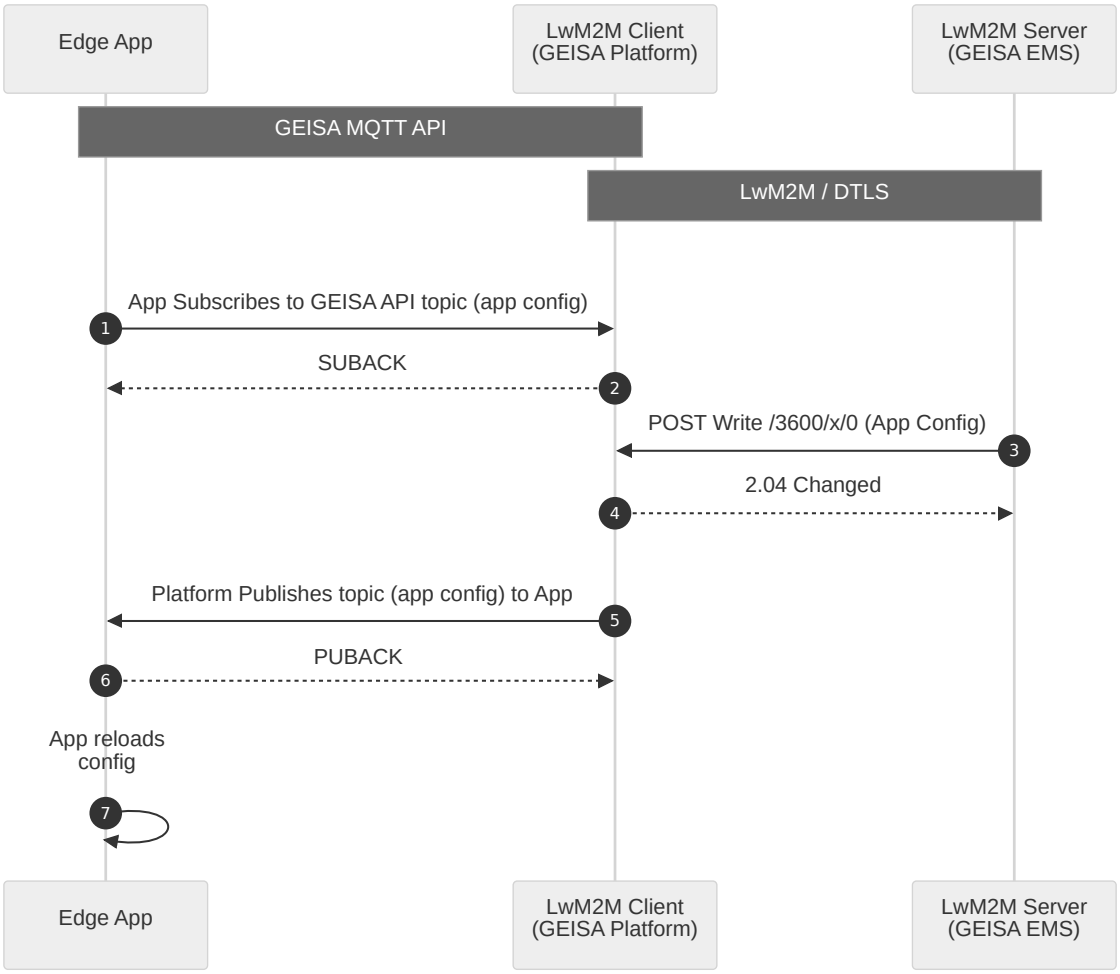
An ADM compliant EMS MAY *Observe* resource 1 to receive asynchronous LwM2M Notifications (containing app data) whenever the platform writes to resource 1 on behalf of the edge app.

An ADM compliant platform will receive asynchronous callbacks from its LwM2M Client whenever the EMS writes to resource 0 to reconfigure an edge app, and the platform SHALL forward that resource 0 payload (containing an app configuration) via GEISA API to the app identified in resource 4050.

Resource ID	Name	Operation	Data Type	Description
0	Server Message	Write	Opaque	Server to Client message used for edge app configuration.
1	Client Message	Read	Opaque	Client to Server message used for edge app data reporting.
2	Message Priority	Read-Write	Integer	Priority: 0=Immediate, 1=BestEffort, 2=Latest
3	Message Desc	Read-Write	Integer	Description: 0=AppConfiguration, 1=AppData
4050	AppID	Read-Write	Integer	ID of the edge app using this object instance.



Edge App Data Reporting



Edge App Configuration

13 Linux Execution Environment

The Linux Execution Environment is one of two execution environments defined within the GEISA specification. The LEE defines requirements for platform implementers that ensure application developers can develop software without needing to customize it for each target environment.

LEE conformant platforms will provide a consistent:

- *Operating System*
- *Application Isolation*
- *Linux Base Libraries*
- *Core Services*
- *Base Filesystem*

13.1 Operating System

GEISA LEE SHALL use Linux as the core operating system. The LEE platform implementer SHOULD use a Linux 6.x or greater kernel. A LEE platform may use an older version if needed, but it MUST provide the needed application isolation requirements as defined within the GEISA specification.

The attack surface and size of the Linux kernel and base libraries provided MUST be minimized. A LEE platform implementation SHOULD remove unnecessary components and libraries providing

the platform and applications the required functionalities while being cognizant and intentional on minimizing the overall packages and services and should not be similar to a full typical Linux distribution.

There is no requirement for the underlying Linux kernel to support real-time features, and no real-time features are exposed to GEISA applications.

Applications, however, do specify their needed CPU and memory resource requirements in their Application manifests and operators SHOULD consider the finite resources available when deploying Applications. Applications SHOULD receive, over time, their resource allotment and not be starved to the point where metrological and waveform data is significantly delayed or lost.

13.2 Application Isolation

Linux Execution Environments are expected to use the containerization capabilites provided natively within the Linux Kernel via cgroups. GEISA does not specify which container engine or management system a platform should use, but it does specify how app images are provided. Platform implementations may use any engine or management system they choose, but they MUST meet the requirements described in this chapter, they MUST provide a base image, and they MUST accept application images.

While platforms may use any container mechanism they choose, for clarity of intent, this portion of the GEISA specification uses `lxc` for various examples.

13.2.1 Isolation Requirements

General Requirements

GEISA applications SHALL be isolated from each other as described in *System Architecture*.

Resource Management

The platform SHALL control every aspect of a GEISA applications access to system resources, including:

- The number of CPUs allocated (e.g. cgroups `limits.cpu`)
- The percentage of CPU allowed (e.g. cgroups `limits.cpu.allowance`)
- The priority of the application (e.g. cgroups `limits.cpu.priority`)
- The maximum RAM (e.g. cgroups `limits.memory`)
- Persistent Storage
- Non-Persistent Storage

Note

Storage limits can be enforced in a variety of ways, depending on the underlying chosen technologies in place. For example, LXC supports the `root size` option. `tmpfs` supports size limits. Quotas may be used. It is up to the platform provider to choose the appropriate technology to enforce limits.

Networking Control

By default, applications are not given any network access. Applications which need relatively simple communication may use the application messaging API that is part of the GEISA API. Apps that require direct access to a network interface may be granted specific permission.

GEISA applications MAY be granted limited network access as described in *System Architecture*.

If granted permissions in the operator manifest, the platform SHALL provide the application with the ability to use standard socket-based IP connectivity from within the containerized environment via the Linux kernel. The platform SHALL be responsible for enforcing the access control list and volume limits described in the application's manifest.

GEISA does not require a specific mechanism for providing network interface access to the containerized environment, however typical options would be a veth pair or passthrough interface using iptables/nftables for policy enforcement.

API Control

Access to the GEISA API is controlled via MQTT permissions. The platform will assign a userid and token (provided to the application in its unique `/etc/geisa/mqtt.conf` file) that the application will use to connect to the message bus. That user will be restricted according to the permissions in the application manifest. Please see *Application Programming Interface (API)* and *Application Manifests* for additional details.

13.2.2 Container Image Requirements

Container images in GEISA are composed by combining a base image (provided by the platform), an application configuration image or directory (provided by the platform), an application image (provided by the application vendor), a non-persistent temporary filesystem or directory (provided by the platform) and a persistent file system or directory (provided by the platform). This combination is mounted as the root file system for the application container. Details of the contents and an example of this file systems can be found in *Base Filesystem*

13.3 Linux Base Libraries

To facilitate clean and regular updates and to minimize container sizes, GEISA applications are encouraged to take advantage of the libraries provided in the base GEISA environment whenever possible.

This does not prevent GEISA applications from including their own libraries in the event that the GEISA environment does not provide a needed library.

There are two major groups of libraries:

- C Language Runtime Libraries (e.g. gcc, uclibc, musl)
- Other C Libraries

The following list is the current MINIMUM expectation for libraries provided in the base GEISA environment. This list is expected to grow over time as the base environment and specification evolves and as applications surface additional common needs.

C Language and Toolchain-provided Runtime Libraries (may vary by implementation):

- libc - Core runtime support
- libgcc - GNU C Compiler Collection (low-level runtime support)
- libstdc++
- libcrypt - Password hashing (MD5, SHA-256)
- libdl - Dynamic loading
- libm - Math library
- libnsl - Network Services Library
- libpthread - POSIX Threads
- libresolv - DNS resolution and name services
- librt - Real-time
- libcap - POSIX capabilities

Other C Libraries (MUST):

- libasyncns - Asynchronous Name Service
- libatomic - Atomic operations
- libcrypto - OpenSSL (hashing, encryption, digital signatures, random numbers, certs/keys)
- libutil - Users, group,s pseudo-ttys (pty), etc.
- libz - compression
- libmosquitto - MQTT client implementation

- libprotobuf - C++ protobuf implementation

Note

Many MQTT client libraries exist and applications MAY use *libmosquitto* listed above or choose a different implementation as needed.

Similarly, applications MAY use *libprotobuf* if their application is C++, otherwise the application must bring their own implementation of protobufs such as but not limited to one listed in Third-Party Add-ons for Protocol Buffers.

The GEISA API is defined at the protocol level and does not mandate a specific application SDK, programming language, or application implementation.

While GEISA ADM  makes use of LWM2M for communication, GEISA Applications are unaware of this and do not require any LWM2M client libraries or knowledge.

13.4 Core Services

Note

Status: Not Yet Defined

The requirements and content for this section are not yet defined in this version of the specification. This section is reserved for future definition in a subsequent specification version.

Implementations SHALL NOT rely on any implied behavior in this section.

13.5 Base Filesystem

A GEISA LEE  container provides the following minimum filesystem within each Application Container environment. The container filesystem contents may be derived from the platform host filesystem or separately constructed specifically for GEISA container environments.

13.5.1 Filesystem Mounts

Each GEISA Application MUST be provisioned in its own container environment separate from other Applications. As such, application-specific names, IDs, and other identifiers do not need to be encoded in filesystem paths within the Application environment.

The GEISA platform implementation MUST provide the following virtual filesystem mounts within the container environment:

- /

  - a filesystem containing base binaries, libraries, the application and other files and directories not specifically listed below. It MUST be either mounted read-only, or the application MUST lack write permission to all items within the filesystem except as explained below.

- /proc
  - a Linux *procfs* filesystem populated by the kernel for the specific application's container
- /dev
  - MAY be part of the / filesystem or separate mount such as *tmpfs* or *devtmpfs* types
  - MUST contain at a minimum: *console full log null random stderr stdin stdout urandom* and *zero*
  - */dev/log* MUST be backed by the platform to filter and direct logs into the GEISA ADM or another system logging mechanism if the implementation is not ADM conformant
- /sys
  - a Linux *sysfs* filesystem populated by the kernel for the specific application's container
- /tmp
  - A separate filesystem or directory within / that is writable by the Application process
  - MUST be limited in size as described in the Application's Deployment Manifest
  - MAY be backed by *tmpfs* or a persistent filesystem
  - Applications MUST handle stale files and cleanup contents it created to remain under limits
  - Platform MAY clean, purge, or delete contents while the Application is NOT running
- /home/geisa
  - A separate filesystem or directory within / that is writable by the Application process
  - MUST be limited in size as described in the Application's Deployment Manifest
  - Applications MUST handle stale files and cleanup contents it created to remain under limits
  - MUST be persistent across application restarts and device reboots
  - Platform SHALL be responsible for metadata integrity (journaling, startup checks, etc…)
  - Application SHALL be tolerant to data corruption and missing files

13.5.2 Utilities and Environment

A base set of executables typically found in an embedded or minimal container environment MUST be present.

GEISA does not require a specific implementation, however Busybox is recommended with the default build options.

A skeleton filesystem MUST be populated including typical paths for binaries and libraries:

- bin
- dev
- etc
- home
- lib
- proc
- sys
- tmp
- var

and optionally:

- sbin
- usr/sbin
- usr/bin
- usr/lib
- run (if present, SHOULD be non-persistent and share */tmp* size limits)

A skeleton filesystem MUST be populated with typical files including at a minimum:

- /etc/group
- /etc/hostname
- /etc/hosts (including a *localhost* entry)
- /etc/passwd

and if provided by libc implementation:

- /etc/resolv.conf
- /etc/nsswitch.conf
- /etc/locale.conf

- /etc/services
- /etc/protocols
- /etc/shells
- /etc/timezone

The following environment variables MUST be set at a minimum when invoking Applications processes:

- SHELL
- HOME
- USER
- PATH

13.5.3 GEISA Components

The Application needs to determine information about the environment it is running on and as such can expect certain GEISA specific configuration to be present. As explained in *API Architecture* and *Platform Discovery* these files are placed in */etc/geisa* within the container environment.

13.5.4 Base Libraries

A set of base libraries SHALL be provided in the base filesystem. See *Linux Base Libraries* for further details.

13.5.5 Construction of the Filesystem

Each Application container filesystem has different content based on the specific Application.

GEISA implementations SHOULD construct a container filesystem using Linux `overlayfs` to reduce flash and ram waste.

Immutable images such as the base utilities and libraries common to all Applications and Application-specific binaries, libraries, and other files provided by the Application vendor SHOULD be `overlayfs lower` layers, while the `upper` MAY be used for generated configuration files, any unix domain sockets and/or the non-persistent storage depending on the design choices of the LEE.

An efficient read-only file system, like `squashfs` for the base and application images SHOULD be used, but this is not required.

Overlayfs allows an Application to provide their own or even replace a base library, fixed data, and executables as needed without having to construct a copy of the lower layer data in flash or ram.

When an Application is upgraded, its overlayfs MUST be re-constructed and any non-persistent files deleted while the persistant files (in */home/geisa*) MUST be preserved.

When an Application is stopped and restarted (or device rebooted), the platform MAY re-construct the filesystem including deletion of any non-persistent files while the persistant files (in */home/geisa*) MUST be preserved.

13.5.6 Example Filesystem Construction

As an example of how this might be composed, consider a platform using `lxc` as its container engine. `/platform` is persistant storage for the GEISA platform and a shared base image is at `/platform/base/geisa-base-1.0.0.sqfs`.

The Platform receives an application manifest for a given application, including the image's digital signature. Upon receiving the application image through the application download process, the platform stores the application image in `/platform/apps/geisa-app-1/app-1.0.1.sqfs`, and then validates the image against the digital signature contained in the manifest.

Based on the permissions specified in the manifest, the platform creates the necessary MQTT user for the application, granting that user access to the necessary APIs allowed in the manifest. It generates a random login token for the application and stores the username and token as the application's user credentials in `/platform/apps/geisa-app-1/config/etc/geisa/mqtt.conf`.

To launch the application the platform will mount each of the required file system images. This example LEE implementation uses a separate writable `/tmp` filesystem mount with the rootfs mounted read-only.

For a system running `lxc`, the platform might create a `geisa-app-1` directory under `/var/lib/lxc`. Then sub-directories for the various `overlayfs` layers and a resulting `rootfs` directory.

The application persistant storage is in `/platform/apps/geisa-app-1/persist` and this example LEE uses a quota to limit application usage as specified in the deployment manifest. Other mechanisms could be used to limit a persistent volume such as a file-backed loopback device.

Example geisa-app-1 with 50MiB persistent and 4MiB non-persistant volumes

```
mkdir -p /platform/apps/geisa-app-1/config/etc/geisa
geisa_create_mqtt_conf geisa-app-1 /platform/apps/geisa-app-1/config/etc/
↪geisa/mqtt.conf

mkdir -p /platform/apps/geisa-app-1/persist
chown geisa-app-1:nogroup /platform/apps/geisa-app-1/persist
chmod 0700 /platform/apps/geisa-app-1/persist
setquota -u geisa-app-1 50M 50M 0 0 /platform

mkdir -p /var/lxc/geisa-app-1/{base,app,config,upper,work,rootfs}
mount -t squashfs /platform/base/geisa-base-1.0.0.sqfs /var/lxc/geisa-app-
↪1/base
mount -t squashfs -o nosuid,nodev /platform/apps/geisa-app-1/app-1.0.1.
↪sqfs /var/lxc/geisa-app-1/app
```

(continues on next page)

(continued from previous page)

```
mount --bind /platform/apps/geisa-app-1/config /var/lxc/geisa-app-1/config

mount -t overlay -oro overlay \
-olowerdir=/var/lxc/geisa-app-1/base:/var/lxc/geisa-app-1/app:/var/lxc/
 ↪geisa-app-1/config \
-oupperdir=/var/lxc/geisa-app-1/upper \
-oworkdir=/var/lxc/geisa-app-1/work \
/var/lxc/geisa-app-1/rootfs

mount --bind /platform/apps/geisa-app-1/persist /var/lxc/geisa-app-1/
 ↪rootfs/home/geisa
mount -t tmpfs -osize=4M,mode=1777 tmpfs /var/lxc/geisa-app-1/rootfs/tmp
mount -t sysfs sysfs /var/lxc/geisa-app-1/rootfs/sys
mount -t proc proc /var/lxc/geisa-app-1/rootfs/proc
```

Note

GEISA LEE SHOULD have their / filesystem mounted read-only in the kernel to follow the principle of least privilege. This prevents Applications from modifying or adding files in unexpected places and forces deterministic Application behavior on each startup.

If an implementation chooses to mount / read-write, it MUST enforce file and directory permissions appropriately as well as limit the growable size of the filesystem to the same limits as the Application's Deployment Manifest specifies for non-persistent storage. In this case a seperate */tmp* mount is unnecessary and any changes outside of the persistent */home/geisa* are non-persistent.

14 Virtual Execution Environment

A Virtual Execution Environment (VEE) is a multi-sandboxed application container for resource-constrained embedded devices running on microcontrollers or microprocessors. It allows devices to run multiple and mixed managed code (Java ® compiled binary code, C/C++ compiled binary code, JavaScript code, etc.).

A VEE is always based on a virtual execution engine that executes inside the device operating system as a process or a task, thus creating an isolated environment where code executes as virtual instructions independent from the operating system or the processor instruction set.

The GEISA Virtual Execution Environment is one of two execution environments defined within the GEISA specification. The GEISA Virtual Execution Environment definitions allows platforms which support running applications in a virtual execution environment, rather than a full operating system, to do so in an interoperable way.

VEEs are used on a variety of platforms. The GEISA VEE may run on top the GEISA Linux Execution Environment (LEE) (refer to *GEISA Linux Execution Environment (LEE)*), although this is not an obligation; that is, the underlying system for a GEISA VEE can be any RTOS-like variant and not necessarily Linux or the GEISA LEE.

VEEs rely on managed-code virtual runtime (typically a virtual machine): GEISA VEE MUST support both multi-thread managed C/C++ and (managed) Java ®. Support for other languages (e.g. Kotlin, Lua, Rust, ECMAScript, etc.) may be included in the future, but is not defined or mandated at this time.

In this version of the GEISA specification the host operating system is Linux and the VEE SHOULD execute in user space as a process. In future versions, it may become possible to consider options such as Zephyr OS, in which case the VEE would be executed as a task.

On top of this runtime sits a multi-application kernel that manages the lifecycle, scheduling, and

isolation of multiple apps running concurrently. Each application executes inside its own sandbox with strict memory and API boundaries, ensuring strong fault-containment, secure separation of logic, and safe coexistence of third-party or field-updatable applications on the same device.

VEE conformant platforms will provide a consistent:

- *Virtual Execution Runtime*
- *Virtual Base Libraries*

14.1 Virtual Execution Runtime

The VEE Runtime is composed of a lightweight execution engine that abstracts the underlying hardware and operating system to provide a trusted and secure, multi-application environment for embedded applications to run on cost-effective devices.

14.1.1 Processing Unit

The GEISA VEE Runtime MUST feature a Processing Unit, which acts as a virtual CPU executing a compact and hardware-agnostic virtual instruction set.

Instead of running native machine code directly on the microcontroller or microprocessor, applications are compiled into optimized virtual bytecode. This bytecode is then executed by the Processing Unit using a deterministic, low-latency execution model tailored for embedded systems.

Because the instruction set is independent from the underlying processor architecture (ARM, RISC-V, x86, etc.) and and also independent from the operating system (Linux, Zephyr, bare metal), the Processing Unit provides a stable abstraction layer that guarantees the same behavior for any binary application across any physical GEISA devices.

14.1.2 Scheduler

Along with the Processing Unit, the VEE Runtime MUST have a strict pre-emptive priority-based scheduler with consistent thread management, ensuring that various applications collaborate and execute with the same behavior on every target, independent of the underlying OS/RTOS.

The GEISA VEE scheduler MUST provide:

- Application-level thread scheduling independent of OS scheduling
- Fair resource allocation among running sandboxed applications
- Prevention of thread starvation across applications
- Consistent scheduling behavior across heterogeneous hardware platforms

14.1.3 Java Language Support

A GEISA conformant VEE MUST support the Java language specification version 7 or later.

The GEISA VEE MUST implement:

- Core Language Features
  - All Java primitives (byte, short, int, long, float, double, boolean, char)
  - Object-oriented features (classes, interfaces, inheritance, polymorphism)
  - Exception handling (try, catch, finally, throw)
  - Reflection API for class inspection and instantiation (Class.forName(), Class.getName(), Class.getSimpleName(), and Class.newInstance())
  - Generic types
- Garbage Collector
  - The garbage collector MAY implement various collection strategies (generational, concurrent, incremental) provided they maintain predictable embedded system behavior.
  - The VEE MUST include a garbage collector to automatically manage memory. The garbage collector MUST provide:
    * Deterministic collection of unreachable objects
    * Minimal impact on application latency suitable for embedded devices
    * Configurable collection policies per sandbox to respect resource constraints

14.1.4 C/C++ Language Support

A GEISA VEE MUST support the C and C++ programming languages. Applications MAY be written entirely in C/C++, or mixed-language applications MAY combine Java and C/C++ components within the same application.

The VEE runtime MUST prevent C/C++ code from bypassing sandbox isolation, accessing other applications' memory, or circumventing security controls through unsafe pointer operations or direct hardware access.

14.1.5 Multi-Sandbox Execution Control

The VEE Runtime MUST include a Multi-Sandbox semantic that provides a dynamic loader for application loading, while enforcing strict isolation between applications at runtime.

The GEISA VEE MUST enforce resource limits per resource (using a Resource Manager):

- Compute Resources: CPU allocation with fair scheduling among running applications, and criticality level between applications
- Memory Resources: Heap size limits per application

- Network Resources: Bandwidth throttling and volume limits

The GEISA VEE MUST provide a Kernel responsible for the minimal core services organized and accessible through a registry of Services. The multi-sandboxed Kernel manages the lifecycle of applications, enforces access control to resources through a Security Manager. The Kernel is responsible for reliable, trusted execution and cannot be modified by applications.

Each application is optional, potentially untrusted, and may be unreliable; however, applications can be installed, started, stopped, and uninstalled without jeopardizing the kernel's stability or the execution of other applications.

The Kernel controls the life cycle of the various loaded applications. An application can be installed, started, stopped and uninstalled at any time independent of the system state (particularly, an application is never allowed to depend on another Application to be stopped).

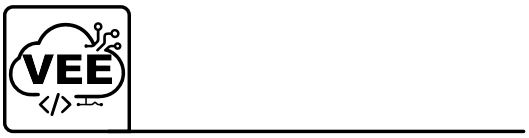
The multi-sandboxed kernel MUST implement the application lifecycle as described below:

- Installed: The Application has been successfully linked to the kernel, but is not running.
- Started: The Application has been started and is running.
- Stopped: The Application has been stopped and all threads are terminated.
- Uninstalled: The application has been unlinked from the Kernel.

14.1.6 Security Manager

A security policy allows the Kernel to prevent an Application from accessing resources or calling specific APIs. A GEISA-conformant VEE multi-sandboxed kernel MUST implement a security manager.

14.2 Virtual Base Libraries



GEISA does not specify which virtual runtime to use for the managed-code memory-safe languages.

For Java ®, a GEISA conformant VEE MUST include support for the following class libraries:

- java.io
- java.lang
- java.lang.annotation
- java.lang.ref

- java.lang.reflect
- java.util
- org.eclipse.paho.client.mqttv5
- com.google.protobuf

Additional libraries may be provided by platforms if desired.

For C and C++, a GEISA conformant VEE MUST provide support for standard libc C library with the exception of OS-specific or non-sandbox-friendly functions such as `fork()/exec()`, process control, and signals.

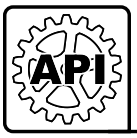
The following APIs MUST be supported:

- **FileSystem:** `open`, `fopen`, `fdopendir`, `close`, `fclose`, `read`, `write`, `fread`, `fwrite`, `lseek`, `stat`, `fstat`, `readdir`, `fstatat`, `fileno`, `fflush`, `access`
- **Sockets:** `send`, `recv`, `shutdown`, `close`, `socket`, `bind`, `listen`, `connect`
- **Clocks:** `clock_getres`, `clock_gettime`, `gettimeofday`
- **Process:** `exit`
- **StdIO:** `printf`
- **Pthreads:** `pthread_create`, `pthread_join`, `pthread_attr_init`, `pthread_attr_setstacksize`, `pthread_attr_destroy`
- **Semaphores:** `sem_wait`, `sem_post`, `sem_destroy`
- **C++ Standard Library**

Note

While GEISA ADM makes use of LWM2M for communication, GEISA Applications are unaware of this and do not require any LWM2M client libraries or knowledge.

15 Application Programming Interface (API)



For GEISA applications to be interoperable across multiple hardware/device platforms, the GEISA specification establishes a well-defined set of APIs between the application and the device-specific portions of platform implementation. *API Architecture* discusses the mechanics of the API.

API Catalog Reference provides a simple catalog of all current APIs.

The GEISA API defines APIs for the following purposes:

- *Platform Discovery*

  Provides platform capabilities, including device type and available features (e.g., sensors, actuators, interfaces), etc.

- *Platform and App Status*

  Provides the current status of the platform, including runtime mode (e.g. Manufacturing, Unprovisioned, Normal, Test), communications status, power status (powered, outage), and resource availability.

- *Instantaneous Data*

  Provides streaming metrology data at one-second intervals or better for applications that need to continuously monitor conditions, without consuming and analyzing waveform data directly.

- Billing Data

  Common instantaneous billing/metrology quantities currently appear in *Instantaneous Data*. A dedicated billing/metered quantities transaction set is reserved for a future version of the specification.

> **Note**
>
> Reserved for expansion and definition in a future version of the specification.

- *Off-Device Communication*

  Allows applications to communicate off-device via messaging or direct IP sockets.

- *Waveform Data*

  Allows applications to obtain waveform frame metadata and to subscribe or unsubscribe to waveform data streams.

- Other Metering Data

  This includes specialized metering quantities that meters may record and that applications may request, but that are not so widely used as to be included in the Instantaneous Data transaction definition.

> **Note**
>
> Reserved for expansion and definition in a future version of the specification.

- *Actuator Status & Control*

  Allows authorized applications to query actuator status and request actuator control for platform-exposed actuators such as service switches, DER switches, and load-control relays.

- *Sensors*

  Exposes sensor data such as temperature, humidity, location, seismic data, vibration, acceleration, and light / solar radiation. These are examples only; sensors are device/platform specific and other sensors may exist on a particular platform.

- Event Data

> **Note**
>
> Reserved for expansion and definition in a future version of the specification.

- System Messaging

> **Note**
>
> Reserved for expansion and definition in a future version of the specification.

Note

Additional transaction sets may be added in future versions of the GEISA specification.

15.1 API Architecture

Rather than provide an interface library, and in the process potentially restricting or specifying which languages are suitable for use with GEISA, GEISA intentionally provides a message bus API allowing for additional implementor flexibility.

15.1.1 Message bus communication

GEISA uses MQTT as its reference message bus. A GEISA platform SHALL provide a MQTT broker and daemon(s) that GEISA applications interact with. GEISA applications MAY interact with the GEISA MQTT broker using the MQTT client library that best suits their application's programming language and architecture. MQTT client libraries exist for many programming languages and developers MAY make use of one provided by the GEISA EE (see *Linux Base Libraries* and *Virtual Base Libraries*) or bundle their own with their application.

APIs are either Request/Response (R/R) type messages from Application to Platform and back, or published data from the Platform to all interested Applications. The architecture of the platform daemons that both respond to R/R messages and post periodic and event data to the MQTT broker is platform-dependent.

GEISA API conformant platforms SHALL:

- provide an MQTT broker version 5 or later.
- support all API transactions listed as mandatory within this specification
- use Protocol Buffers (syntax=proto3) for message payloads
- use the MQTT topics indicated for the API transactions
- support the Protobuf definition indicated for the API transactions found at https://github.com/geisa/schemas

The GEISA API is designed as an **internal** API within the platform. It is not intended to be exposed to a general network environment. For security reasons, platform implementations MUST not expose the GEISA API MQTT implementation via external network interfaces.

Note

Platforms or applications running in a GEISA-complaint COE MAY choose to provide external MQTT implementations separate from the GEISA API MQTT implementation.

15.1.2 Message bus connection and credentials

The GEISA API SHALL use MQTT standard port 1883. GEISA API transactions are not sent over external network interfaces or shared network interfaces. GEISA API transactions SHALL NOT require the use of TLS.

The GEISA API MQTT implementation may only be exposed via `localhost` or via virtual interfaces to GEISA application containers.

To enable authentication, GEISA EE  conformant applications SHALL be assigned a unique user ID and token by the platform. Applications use the credentials to authenticate with the MQTT message broker providing the GEISA API.

The platform SHALL generate and manage application user ID and tokens locally on-device. User IDs and tokens MUST be alphanumeric printable strings. Tokens MUST contain random data of sufficient length. User ID values persist for the life of the application's installation and tokens persist for at least the life of an application's execution.

Note

It is possible for a platform to use other means to determine which application instance is connecting to the message broker and pass authorization information out-of-band, and skip authentication entirely. Such an approach, while possible, requires platform vendors to customize or otherwise modify the MQTT broker. This simplified approach of passing authentication credentials to the application as part of its local configuration enables the light-weight authentication of applications to the broker to be appropriate for a closed environment, without requiring unwanted customizations. It is understood that this approach is wholly inappropriate in a general network context.

15.1.3 Message bus topics and reliability

API requests SHALL be sent at MQTT QoS level 1, *At least once*, with acknowledged delivery. The GEISA API MQTT broker shall acknowledge API requests. GEISA API responses are sent at the QoS level indicated for the transaction, but most messages are QoS level 0, *At most once* and unacknowledged.

MQTT topics used by the API are listed within each subsequent section.

15.1.4 Non-message bus communication

While most application to platform communication is performed over the MQTT message bus, there are a few exceptions:

In order for an Application to establish a connection to the MQTT message bus, the connection method (host and port) and per-application credentials are provided out-of-band from platform to application. GEISA applications determine how to connect to the broker from this data.

For GEISA LEE, this information SHALL be available at the well-known location `/etc/geisa/mqtt.conf` within the Application container environment.

```
$ cat /etc/geisa/mqtt.conf
HOST=localhost
PORT=1883
USERID=myuserid
PASSWORD=mytoken
```

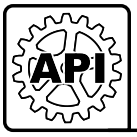
Note

As a reminder, each LEE application is run within its own isolated environment enabling this data to be placed within a fixed well-known location and remain unencrypted accessible only to a single instance of a single application.

For GEISA VEE, this information SHALL be made available to the Application by the platform. The specific mechanism by which it is exposed is not defined and remains TBD in this version of the specification.

Because of the potentially very high-volumes of data involved in accessing metrology *Waveform Data*, it is intentionally handled as a special case. The waveform data transaction allows applications to obtain the frame format and platform specific details regarding the waveform data (e.g. sampling rate, sample resolution, etc.) and to request that the waveform data stream is activated (or deactivated), but the waveform data itself is provided as a raw-binary data structure pushed from the platform to any target application containers using a socket and data format as described in *Waveform Data*.

15.2 API Catalog Reference



This section is not intended to be read as a narrative description of system behavior, but to serve as

a structured reference for developers implementing or integrating with the GEISA API.

It serves as a comprehensive catalog of all defined API calls, including:

- MQTT topic structures
- Required and optional parameters
- Message payload formats
- Expected responses
- Error conditions, where applicable

GEISA APIs are implemented using MQTT topic hierarchies. Each API call is defined by:

- The MQTT topic structure
- The direction of message flow
- The associated request and response payload definitions
- References to relevant sections of the GEISA specification that define relevant behavior and constraints

Note

In the event of a conflict between this API catalog and the specific description in each API reference section, the API reference section SHALL prevail.

15.2.1 API Topic and Permission Catalog

Note

In the topic definitions below, <userid> refers to the platform-local identifier of the requesting GEISA application instance (see *Glossary*).

Table 15.1: API Topic and Permission Catalog

Category	Topic	Role	Action	Direction	Message Name	Message Type
Discovery	geisa/api/platform/discovery/req/<userid>	Application	Publish	App → Platform	GeisaPlatformDiscovery_Req	Request

continues on next page

Table 15.1 – continued from previous page

Category	Topic	Role	Action	Direction	Message Name	Message Type
	geisa/api/platform/discovery/rsp/<userid>	Application	Subscribe	Platform → App	GeisaPlatformI	Response ery_Rsp
	geisa/api/app/manifest/req/<userid>	Application	Publish	App → Platform	GeisaApplicat:	Request loymentManif
	geisa/api/app/manifest/rsp/<userid>	Application	Subscribe	Platform → App	GeisaApplicat:	Response loymentManif
	geisa/api/platform/discovery/req/#	Platform	Wildcard Subscribe	App → Platform	GeisaPlatformI	Request ery_Req
	geisa/api/platform/discovery/rsp/<userid>	Platform	Publish	Platform → App	GeisaPlatformI	Response ery_Rsp
	geisa/api/app/manifest/req/#	Platform	Wildcard Subscribe	App → Platform	GeisaApplicat:	Request loymentManif
	geisa/api/app/manifest/rsp/<userid>	Platform	Publish	Platform → App	GeisaApplicat:	Response loymentManif
Platform Status	geisa/api/platform/status	Application	Subscribe	Platform → App	GeisaPlatformS	Broadcast tatus
	geisa/api/platform/app/status/<userid>	Application	Subscribe	Platform → App	GeisaPlatformS	Directed tatus
	geisa/api/app/platform/status/<userid>	Application	Publish	App → Platform	GeisaAppToPla:	Directed tatus

continues on next page

Table 15.1 – continued from previous page

Category	Topic	Role	Action	Direction	Message Name	Message Type
	`geisa/api/platform/status`	Platform	Publish	Platform → App	`GeisaPlatform`	Broadcast `tatus`
	`geisa/api/platform/app/status/<userid>`	Platform	Publish	Platform → App	`GeisaPlatform`	Directed `tatus`
	`geisa/api/app/platform/status/#`	Platform	Wildcard Subscribe	App → Platform	`GeisaAppToPla`	Directed `tatus`
Networking	`geisa/api/message/upstream/req/<userid>`	Application	Publish	App → Platform	`GeisaAppMessa`	Request
	`geisa/api/message/upstream/rsp/<userid>`	Application	Subscribe	Platform → App	`GeisaAppMessa`	Response
	`geisa/api/message/downstream/req/<userid>`	Application	Subscribe	Platform → App	`GeisaAppMessa`	Request
	`geisa/api/message/downstream/rsp/<userid>`	Application	Publish	App → Platform	`GeisaAppMessa`	Response
	`geisa/api/message/upstream/req/#`	Platform	Wildcard Subscribe	App → Platform	`GeisaAppMessa`	Request
	`geisa/api/message/upstream/rsp/<userid>`	Platform	Publish	Platform → App	`GeisaAppMessa`	Response
	`geisa/api/message/downstream/req/<userid>`	Platform	Publish	Platform → App	`GeisaAppMessa`	Request

continues on next page

Table 15.1 – continued from previous page

Category	Topic	Role	Action	Direction	Message Name	Message Type
	geisa/api/message/downstream/rsp/#	Platform	Wildcard Subscribe	App → Platform	GeisaAppMessa	Response
Instantaneous Data	geisa/api/instantaneous/data	Application	Subscribe	Platform → App	GeisaInstanta	Broadcast
	geisa/api/instantaneous/data	Platform	Publish	Platform → App	GeisaInstanta	Broadcast
Waveform Data	geisa/api/waveform/req/<userid>	Application	Publish	App → Platform	GeisaWaveform_	Request
	geisa/api/waveform/rsp/<userid>	Application	Subscribe	Platform → App	GeisaWaveform_	Response
	geisa/api/waveform/req/#	Platform	Wildcard Subscribe	App → Platform	GeisaWaveform_	Request
	geisa/api/waveform/rsp/<userid>	Platform	Publish	Platform → App	GeisaWaveform_	Response
Actuator Status & Control	geisa/api/actuator/get/req/<userid>	Application	Publish	App → Platform	GeisaActuator(	Request
	geisa/api/actuator/get/rsp/<userid>	Application	Subscribe	Platform → App	GeisaActuator(	Response
	geisa/api/actuator/set/req/<userid>	Application	Publish	App → Platform	GeisaActuator!	Request

continues on next page

Table 15.1 – continued from previous page

Category	Topic	Role	Action	Direction	Message Name	Message Type
	geisa/api/actuator/set/rsp/<userid>	Application	Subscribe	Platform → App	GeisaActuator	Response
	geisa/api/actuator/get/req/#	Platform	Wildcard Subscribe	App → Platform	GeisaActuator	Request
	geisa/api/actuator/get/rsp/<userid>	Platform	Publish	Platform → App	GeisaActuator	Response
	geisa/api/actuator/set/req/#	Platform	Wildcard Subscribe	App → Platform	GeisaActuator	Request
	geisa/api/actuator/set/rsp/<userid>	Platform	Publish	Platform → App	GeisaActuator	Response
Sensors	geisa/api/sensor	Application	Subscribe	Platform → App	GeisaSensorRe	Broadcast
	geisa/api/sensor-req/<userid>	Application	Publish	App → Platform	GeisaSensorRe	Request
	geisa/api/sensor-rsp/<userid>	Application	Subscribe	Platform → App	GeisaSensorRe	Response
	geisa/api/sensor	Platform	Publish	Platform → App	GeisaSensorRe	Broadcast
	geisa/api/sensor-req/#	Platform	Wildcard Subscribe	App → Platform	GeisaSensorRe	Request

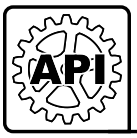
continues on next page

Table 15.1 – continued from previous page

Category	Topic	Role	Action	Direction	Message Name	Message Type
	`geisa/api/sensor-rsp/<userid>`	Platform	Publish	Platform → App	`GeisaSensorRe`	Response `_Rsp`

Actuator payloads are defined by `GeisaActuatorGet_Req`, `GeisaActuatorGet_Rsp`, `GeisaActuatorSet_Req`, and `GeisaActuatorSet_Rsp`. Status and control are permission-gated. Actuator support and target availability are platform-specific.

15.3 Platform Discovery



Different hardware platform implementers may choose to offer different capabilities. With this in mind, the GEISA API provides a mechanism for describing or enumerating the specific hardware platform's capabilities. This platform discovery capability allows the application to discover what a given platform offers in order to potentially adjust behavior, or just for logging and awareness purposes.

The Platform Discovery data is typically constant and is not expected to change during an application's runtime execution. It SHOULD be retrieved by a GEISA application during startup after the application becomes operational. Although the Platform Discovery content is platform-wide rather than application-specific, it is retrieved using application-instance request/response topics so that each running application instance can obtain the current discovery snapshot for its device without broadcasting responses to other applications that may still be subscribed.

After receiving this data, an application MAY unsubscribe from the Platform Discovery response topic for the remainder of that execution instance, as the data will not change during runtime.

Applications MUST re-run Platform Discovery on each startup; if platform capabilities change, the platform will restart applications so they retrieve an updated snapshot during startup.

Dynamic updates and state are provided in *Platform and App Status*.

Platform Discovery metadata may be truly static, where a platform implementer generates the platform metadata as part of a build process. This may be true, for example, for a fixed function device with no installation variants. It is also possible that a platform vendor may dynamically generate the platform metadata at the time of device installation or first boot. This may be true if the GEISA

platform is hosted on an edge card that needs to be married to a device, or a device whose realized capabilities may be activated as part of the installation process. How the platform metadata is generated is outside the scope of GEISA. However, because platform metadata is expected to be static, GEISA platforms MUST facilitate restarting all running applications in the event that the platform metadata changes; applications will be shut down by the platform and subsequently started with the updated Platform Discovery responses, so on startup applications are again aware of any changes/current platform capabilities.

15.3.1 Hardware, Firmware, and Platform Software

The Platform Discovery API includes the following data so applications are able to determine the realized hardware and software environment available on the device:

- Device Type - returns the type of the device or module, e.g. electric meter, EV charger, etc.
- Device Info - returns manufacturer name, model, revision, serial number(s)
- Device Firmware - returns a list of the device platform/OS firmware versions (including notable libraries and firmware and versions)
- Platform Software Environment - returns notable platform software and runtime components and their versions
- Operator Information - if provisioned, the operator name and operator-assigned device identifier. If the device is not provisioned to an operator, the Operator Information MAY be omitted from the Platform Discovery response.

Note

The platform vendor determines what information is included in the Device Firmware information. However, GEISA mandates that it MUST include, at a minimum, the versions of:

1. the base host OS
2. the GEISA API implementation
3. the GEISA platform runtime environment in use, such as LEE or VEE where implemented

15.3.2 Application Information

Because an application's Deployment Manifest may differ from the Vendor Application Manifest originally supplied by the application vendor, applications MUST be able to retrieve their own current Application Deployment Manifest as described in *Application Manifests*.

15.3.3 Metrology Hardware

For Meter type devices, the platform MUST provide at a minimum:

- Meter Rating/Class
- Meter Form
- Number of Phases and if Neutral is connected
- Nominal Phase Angle
- Nominal Frequency
- Nominal Phase-to-Phase and Phase-to-Neutral Voltage (when applicable)

15.3.4 Sensor Hardware

A GEISA EE compliant platform can optionally provide one or more sensors.

Some example sensors that may be provided include:

- Temperature Sensor and Alarm
- Humidity Sensor
- Switch Sensor
- Orientation or Motion/Vibration Sensor
- GNSS

Note this list is not exhaustive, and specific sensor types and accuracy will be device-dependent.

Platform Discovery MAY include a list of available sensor descriptors so that applications can enumerate which sensors exist on the platform at startup. Runtime sensor readings and application-directed sensor queries are defined by the Sensors API (*Sensors*).

15.3.5 Network Hardware

Status: Reserved for Future Definition

Network hardware discovery remains reserved for future definition.

Implementations SHALL NOT assume behavior, interface, or data structure beyond what is explicitly defined elsewhere in this specification.

The Application SHALL assume that network permissions and volume restrictions described in it's Deployment Manifest are valid and realizable on the device

15.3.6 Waveform Discovery

Platform Discovery describes the available waveform streams and their characteristics. Platforms that support waveform data SHALL expose a baseline waveform stream with the identifier `waveform-base`.

Discovery SHALL include only static metadata. Applications SHALL use waveform metadata to interpret a stream and SHALL NOT infer stream characteristics by parsing the stream identifier.

Runtime access details and application-specific state are defined by the *Waveform Data* API.

15.3.7 Application Deployment Manifest

Each GEISA application may retrieve its own Application Deployment Manifest. This manifest is also constant and not expected to change during runtime.

Like the Platform Discovery metadata, the Application Deployment Manifest may be truly static; however, it is also possible that the operator of the platform changes this data at runtime on an infrequent basis.

Because the Application Deployment Manifest content is expected to be static, GEISA platforms MUST ensure a running application is restarted in the event of a manifest change.

Warning

Application designers should consider the impact of placing configuration-type data in their application manifest to minimize the occurences where an operator needs to change the manifest. Frequently updated values or parameters should be sent over another mechanism such as the application message based communication with the ADM system.

MQTT Details

- QoS: 1 / Acknowledged R/R
- Req Topic: `geisa/api/platform/discovery/req/<userid>` and `geisa/api/app/manifest/req/<userid>`
- Rsp Topic: `geisa/api/platform/discovery/rsp/<userid>` and `geisa/api/app/manifest/rsp/<userid>`

Note

In the topic definitions, `<userid>` refers to the platform-local identifier of the requesting GEISA application instance (see *Glossary* for additional information).

The MQTT topics for geisa/api/app/manifest/* are used to retrieve the Application Deployment Manifest for the requesting application instance.

API Permissions

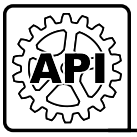
- Application:
  - Publish: `geisa/api/platform/discovery/req/<userid>`
  - Subscribe: `geisa/api/platform/discovery/rsp/<userid>`
  - Publish: `geisa/api/app/manifest/req/<userid>`
  - Subscribe: `geisa/api/app/manifest/rsp/<userid>`
- Platform:
  - Wildcard Subscribe: `geisa/api/platform/discovery/req/#`
  - Publish: `geisa/api/platform/discovery/rsp/<userid>`
  - Wildcard Subscribe: `geisa/api/app/manifest/req/#`
  - Publish: `geisa/api/app/manifest/rsp/<userid>`

Transaction Data

- `GeisaPlatformDiscovery_Req`
- `GeisaPlatformDiscovery_Rsp`
- `GeisaApplicationDeploymentManifest_Req`
- `GeisaApplicationDeploymentManifest_Rsp`

As defined in https://github.com/geisa/schemas

15.4 Platform and App Status



While the *Platform Discovery* data is fixed and does not change during the life of the application, the platform uses the message bus to distribute status to all running applications. Notifications are grouped into 2 topics: application-specific or non-application-specific.

15.4.1 Runtime Mode

Devices may be changed between a normal production operating mode, test modes, and similar states. These states may not apply to a specific GEISA application, but applications can use these notifications to enable or disable their own modes or behaviors, suppress errors and alarms, or connect to different cloud resources (e.g., connect to test vs production service instances). Runtime mode is non-application-specific and determined solely by the device. Runtime mode is platform-scoped and applies uniformly to all applications running on the device.

GEISA defines the following runtime modes:

- Manufacturing
- Unprovisioned
- Normal
- Test

15.4.2 Urgent Platform Updates

If the platform encounters an error or edge condition it can inform GEISA applications so they can take appropriate actions. For some conditions, a GEISA application may be able to directly determine the same error or condition, but a platform notification allows for a consistent method of notification and can reduce duplication of effort between platform and applications.

A GEISA compliant implementation MUST provide at a minimum non-application-specific notifications for:

- Over temperature
- High CPU usage
- Low Memory
- Power Degraded (individual phase loss, under voltage, etc…)
- Power Loss (running on backup power)
- Reboot Scheduled
- Shutdown Scheduled

When an application receives one of these updates it MAY take an action but SHALL continue operating until a separate application shutdown request is received, if any. The platform will send these updates periodically (at least once per minute) for as long as the condition persists. The platform SHOULD send a final notification when the condition clears.

15.4.3 Application Updates

GEISA applications should be made aware of their own resource utilization and any events or alarms or commands specific to their application instance.

A GEISA-compliant implementation MUST provide at a minimum application-specific notifications for:

- Resource usage periodically (at least once every 15 seconds)
  - CPU usage
  - Memory usage
  - Persistent storage usage
  - Non-persistent storage usage
- Control requests for the application:
  - Request application to send status immediately
  - Request application to shut down cleanly
  - Request application to clear PII from memory and storage, if any

The platform MAY request that an application perform a clean shutdown via the message bus, OR it MAY invoke the stop command defined in the application manifest. These mechanisms are considered functionally equivalent from the application's perspective.

A shutdown request delivered via the message bus is advisory, allowing the application to perform an orderly shutdown (e.g., persist state, close connections, flush logs). The platform remains responsible for enforcing termination using the defined stop and termination behavior if the application does not complete shutdown within the allowed time.

15.4.4 Timestamps

All platform-to-application status messages SHALL include a timestamp representing the time at which the message was generated by the platform.

The timestamp SHALL be expressed as UTC epoch time in milliseconds.

Applications MAY use this timestamp for ordering, correlation, and time-based calculations relative to device-local policy such as daily resource reset intervals.

15.4.5 Application Status

GEISA platforms require visibility into the status and health of each application in order to manage lifecycle and availability.

A GEISA-compliant implementation MUST accept at a minimum application-specific messages for:

- Notification that an application startup is complete and now operating
- Current application status (running, shutting down)
- Notification that an application is shut down (due to a shutdown request) and ready for termination
- Request from an application to terminate itself then restart
- Request from an application to terminate itself without restart

Upon startup of an application, the application MUST send a notification to the platform that it is now operating and its status is RUNNING. In this message, the application MAY indicate whether it expects the platform to provide a keepalive/watchdog mechanism. If so, the application SHALL also send its application status periodically (at a period specified in the most recent message). If the application does not require a keepalive/watchdog mechanism, it MAY omit a timeout value.

If the platform does not receive a status message within the specified timeout period it MUST take the following corrective actions:

- Send a message to the application requesting its status
- If the application does not respond within a second timeout period, run the stop command specified in the application manifest
- If the stop command itself times out (as defined in the application manifest), terminate the application
- Restart the application using the start command as defined in the application manifest

15.4.6 Connectivity Updates

A GEISA compliant implementation MUST provide at a minimum non-application-specific notifications for:

- Message-based communication:
  - status: disabled, enabled-down, or enabled-up
- Operator network IP communication:
  - status: disabled, enabled-down, or enabled-up
  - protocol: ipv4, ipv6, or both
  - latency: real-time, delayed
- Internet IP communication:
  - status: disabled, enabled-down, or enabled-up
  - protocol: ipv4, ipv6, or both
  - latency: real-time, delayed

- Local network communication:
  - status: disabled, enabled-down, or enabled-up
  - protocol: ipv4, ipv6, or both
  - latency: real-time, delayed

A GEISA compliant implementation MUST provide at a minimum application-specific notifications for:

- Time of next daily volume reset
- Message based communication:
  - Previous day messages used
  - Daily messages used & remaining
- Operator based IP communication:
  - Volume state: zero, metered, unlimited
  - Previous day volume used while unlimited
  - Previous day volume used while metered
  - Daily volume used while unlimited
  - Daily volume used & remaining while metered
- Internet based IP communication:
  - Volume state: zero, metered, unlimited
  - Previous day volume used while unlimited
  - Previous day volume used while metered
  - Daily volume used while unlimited
  - Daily volume used & remaining while metered
- Local based IP communication:
  - Volume state: zero, metered, unlimited
  - Previous day volume used while unlimited
  - Previous day volume used while metered
  - Daily volume used while unlimited
  - Daily volume used & remaining while metered

MQTT Details

- QoS: 0 / Unacknowledged
- Topic: `geisa/api/platform/status`
- Topic: `geisa/api/platform/app/status/<userid>`
- Topic: `geisa/api/app/platform/status/<userid>`

Note

In the topic definitions, `<userid>` refers to the platform-local identifier of the requesting GEISA application instance (see *Glossary* for additional information).

API Permissions

- Application:
  - Subscribe: `geisa/api/platform/status`
  - Subscribe: `geisa/api/platform/app/status/<userid>`
  - Publish: `geisa/api/app/platform/status/<userid>`
- Platform:
  - Publish: `geisa/api/platform/status`
  - Publish: `geisa/api/platform/app/status/<userid>`
  - Subscribe: `geisa/api/app/platform/status/#`

Transaction Data

- `GeisaPlatformToAppStatus`
- `GeisaAppToPlatformStatus`

as defined in https://github.com/geisa/schemas

15.5 Instantaneous Data

15.5.1 Summary

Instantaneous data provides a continuous stream of metrological data as provided by the host device, with data being pushed at least once a second. Platforms MAY push data more frequently; however, data push periodicity MUST be consistent. The instantanous data push periodicity MUST be reported in the *Platform Discovery* transaction.

Instantaneous data is always available to authorized applications. There is no need to specifically request or activate it; applications can simply subscribe to the topic.

To enable application developers to have a consistent environment, the instantaneous data message includes many mandatory measurements.

Simultaneously, GEISA recognizes that data which may be appropriate to require from every AC electric meter may be problematic to obtain from DC meters or fault indicators, load tap changers or other devices. To address this, GEISA uses the concept of a *device type*. The device type must be reported as part of the *Platform Discovery*.

The specific values required to be published by a GEISA API conformant platform will depend on the device type. Presently, GEISA offers two device types: AC electric meter and unspecified. Additional device types may be defined in the future.

Table 15.2: Instantaneous Data

ID	Description	Unit of Measure	Data Type	Details	AC Polyphase Meter	AC Meter	Unspecified	Comments
0	UTC Time	milliseconds	int-64	Since 1970/1/1 00:00 UTC	M	M	M	
1	RMS Voltage Phase A	Volts	float-64	Average	M	M	O	Used as system voltage on split phase systems
2	RMS Current Phase A	Amps	float-64	Average	M	M	O	
3	RMS Voltage Phase B	Volts	float-64	Average	M	O	O	
4	RMS Current Phase B	Amps	float-64	Average	M	O	O	

continues on next page

Table 15.2 – continued from previous page

ID	Description	Unit of Measure	Data Type	Details	AC Polyphase Meter	AC Meter	Unspecified	Comments
5	RMS Voltage Phase C	Volts	float-64	Average	M	O	O	
6	RMS Current Phase C	Amps	float-64	Average	M	O	O	
7	RMS Current Neutral	Amps	float-64	Average	O	O	O	
8	Watts Delivered Phase A	Watts	float-64		M	M	O	
9	Watts Received Phase A	Watts	float-64		M	M	O	
10	VAR Delivered Phase A	Volt-Ampere Reactive	float-64		M	M	O	
11	VAR Received Phase A	Volt-Ampere Reactive	float-64		M	M	O	
12	VA Q1 Phase A	Volt-Ampere	float-64		M	M	O	
13	VA Q2 Phase A	Volt-Ampere	float-64		M	M	O	
14	VA Q3 Phase A	Volt-Ampere	float-64		M	M	O	
15	VA Q4 Phase A	Volt-Ampere	float-64		M	M	O	
16	Watts Delivered Phase B	Watts	float-64		M	O	O	

continues on next page

Table 15.2 – continued from previous page

ID	Description	Unit of Measure	Data Type	Details	AC Polyphase Meter	AC Meter	Unspecified	Comments
17	Watts Received Phase B	Watts	float-64		M	O	O	
18	VAR Delivered Phase B	Volt-Ampere Reactive	float-64		M	O	O	
19	VAR Received Phase B	Volt-Ampere Reactive	float-64		M	O	O	
20	VA Q1 Phase B	Volt-Ampere	float-64		M	O	O	
21	VA Q2 Phase B	Volt-Ampere	float-64		M	O	O	
22	VA Q3 Phase B	Volt-Ampere	float-64		M	O	O	
23	VA Q4 Phase B	Volt-Ampere	float-64		M	O	O	
24	Watts Delivered Phase C	Watts	float-64		M	O	O	
25	Watts Received Phase C	Watts	float-64		M	O	O	
26	VAR Delivered Phase C	Volt-Ampere Reactive	float-64		M	O	O	
27	VAR Received Phase C	Volt-Ampere Reactive	float-64		M	O	O	
28	VA Q1 Phase C	Volt-Ampere	float-64		M	O	O	
29	VA Q2 Phase C	Volt-Ampere	float-64		M	O	O	

continues on next page

Table 15.2 – continued from previous page

ID	Description	Unit of Measure	Data Type	Details	AC Polyphase Meter	AC Meter	Unspecified	Comments
30	VA Q3 Phase C	Volt-Ampere	float-64		M	O	O	
31	VA Q4 Phase C	Volt-Ampere	float-64		M	O	O	
32	Watts Delivered Phase System	Watts	float-64		M	O	O	
33	Watts Received Phase System	Watts	float-64		M	O	O	
34	VAR Delivered Phase System	Volt-Ampere Reactive	float-64		M	O	O	
35	VAR Received Phase System	Volt-Ampere Reactive	float-64		M	O	O	
36	VA Q1 Phase System	Volt-Ampere	float-64		M	O	O	
37	VA Q2 Phase System	Volt-Ampere	float-64		M	O	O	
38	VA Q3 Phase System	Volt-Ampere	float-64		M	O	O	
39	VA Q4 Phase System	Volt-Ampere	float-64		M	O	O	
40	Phase A Voltage Angle	Degrees	float-32		O	O	O	

continues on next page

Table 15.2 – continued from previous page

ID	Description	Unit of Measure	Data Type	Details	AC Polyphase Meter	AC Meter	Unspecified	Comments
41	Phase A Current Angle	Degrees	float-32		O	O	O	
42	Phase A Power Factor	Unitless Ratio	float-32		O	O	O	
43	Phase A Power Factor Angle	Degrees	float-32		O	O	O	
44	THD Phase A Voltage	Unitless Ratio	float-32		O	O	O	
45	THD Phase A Current	Unitless Ratio	float-32		O	O	O	
46	Phase B Voltage Angle	Degrees	float-32		O	O	O	
47	Phase B Current Angle	Degrees	float-32		O	O	O	
48	Phase B Power Factor	Unitless Ratio	float-32		O	O	O	
49	Phase B Power Factor Angle	Degrees	float-32		O	O	O	
50	THD Phase B Voltage	Unitless Ratio	float-32		O	O	O	
51	THD Phase B Current	Unitless Ratio	float-32		O	O	O	
52	Phase C Voltage Angle	Degrees	float-32		O	O	O	

continues on next page

Table 15.2 – continued from previous page

ID	Description	Unit of Measure	Data Type	Details	AC Polyphase Meter	AC Meter	Unspecified	Comments
53	Phase C Current Angle	Degrees	float-32		O	O	O	
54	Phase C Power Factor	Unitless Ratio	float-32		O	O	O	
55	Phase C Power Factor Angle	Degrees	float-32		O	O	O	
56	THD Phase C Voltage	Unitless Ratio	float-32		O	O	O	
57	THD Phase C Current	Unitless Ratio	float-32		O	O	O	

Note

Some jurisdictions such as Canada, prefer to use vectorial calculations rather than arithmetic calculations for generating metered quantities. Rather than having the instantaneous data feed provide both, a single stream of data is offered. Whether the platform is providing arithmetic or vectorial values is indicated by the *Platform Discovery* transaction.

15.5.2 MQTT Details

- QoS: 0 / Unacknowledged
- Topic: `geisa/api/instantaneous/data`

15.5.3 API Permissions

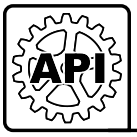
- Application:
  - Subscribe: `geisa/api/instantaneous/data`
- Platform:
  - Publish: `geisa/api/instantaneous/data`

15.5.4 Transaction Data

- `GeisaInstantaneousQuantities`

As defined in https://github.com/geisa/schemas

15.6 Waveform Data



The GEISA Waveform Data API provides access to higher-volume waveform data such as voltage and current samples. Because of the potentially high data rates involved, waveform data is handled differently than lower-rate APIs such as sensors. The request/response messages exchanged over MQTT are used to request access to a waveform stream and to provide the metadata needed to decode the returned waveform frames. The waveform samples themselves are not transported as MQTT payloads.

Applications publish waveform requests to `geisa/api/waveform/req/<userid>` and subscribe for responses on `geisa/api/waveform/rsp/<userid>`. On a successful subscribe request, the platform returns a per-application `AF_UNIX SOCK_SEQPACKET` socket path. The application then consumes waveform frames from that socket.

Applications are not expected to infer electrical topology, meter form, or device classification from waveform stream identifiers. Stream identifiers are used to select among the waveform streams offered by the platform. Device Type, metrology hardware characteristics, meter form, number of phases, nominal frequency, and similar platform-wide details are provided through *Platform Discovery*,

15.6.1 Baseline Stream Requirement

Platforms that support waveform data SHALL expose a baseline waveform stream with the identifier `waveform-base`.

The `waveform-base` stream represents the baseline interoperable waveform stream for GEISA applications. Platforms MAY expose additional waveform streams with different identifiers to provide alternate representations such as reduced, decimated, filtered, ultra-high frequency, raw, or otherwise specialized waveform data.

Applications SHALL use waveform metadata to determine stream characteristics. Applications SHALL NOT infer waveform structure or platform characteristics by parsing the `stream-id` value.

If present, waveform stream `name` and `description` values are platform-supplied informational metadata intended for display, diagnostics, or logging. Applications SHALL NOT rely on these

fields for interoperability behavior.

15.6.2 Device Capabilities

A platform MAY provide zero or more waveform streams. When waveform data is supported, *Platform Discovery* SHALL indicate the presence of waveform support and SHALL expose the descriptor metadata for each available waveform stream.

A waveform stream descriptor SHALL include:

- A platform-defined `stream-id`.
- Optional human-readable `name` and `description` fields.
- The waveform `sample-type` one of `int16`, `int32`, `float32`, or `float64`.
- The number of voltage channels present in each sample index.
- The number of current channels present in each sample index.
- The total number of channels present in each sample index. This SHALL equal voltage-channel-count plus current-channel-count.
- The waveform `sample-rate-hz`.
- The waveform `samples-per-cycle` when applicable.
- The waveform `nominal-frequency-hz` when applicable.
- Whether frames are `cycle-aligned`.
- Whether frames are `zero-crossing-aligned`.
- Optional integer-sample `voltage-scale` and `current-scale` values.
- The expected frame delivery period in milliseconds.

Platforms that provide waveform data SHALL provide, at a minimum:

- 128 or more samples per nominal AC cycle where AC cycle semantics apply.
- 16-bit or more sample resolution.
- Frame delivery at least every 200 ms for the baseline stream.
- For zero-crossing-aligned polyphase data, alignment to the Phase A voltage zero crossing.

Different devices will have different numbers of capture channels, each typically sampling a single voltage or current input. The total number of channels and how those channels are assigned may not be known at application build time and are discoverable through *Platform Discovery* metadata.

Data for all channels in a frame SHALL be time-aligned to one another. An individual sample in one channel SHALL have been sampled at the same time as the corresponding samples in other channels for that sample index.

Note

Platforms MAY expose multiple waveform stream choices to applications if they support multiple capture frequencies, channel counts, filters, or alternate waveform views. In this case, an application MAY subscribe to one or more streams as permitted by the platform and application deployment manifest.

Platform Discovery SHALL expose only static waveform descriptor metadata. Runtime access handles such as socket paths SHALL NOT be included in *Platform Discovery*.

15.6.3 API Request Response

A waveform request includes:

- `stream-id` – the identifier of the waveform stream.
- `request-type` – either `subscribe` or `unsubscribe`.

A waveform response includes:

- `status` – success or a defined error condition.
- `stream-id` – echo of the requested stream identifier.
- `subscribed` – whether the application instance is subscribed after the request is processed.
- `socket-path` – the returned per-application socket path when the application is subscribed.
- The waveform metadata needed to decode the returned frames.

A waveform subscribe request does not imply that an application controls the underlying platform metrology hardware or the global lifecycle of waveform capture. Rather, the request asks the platform to grant or deny access to a waveform stream for the requesting application instance. Multiple applications MAY consume waveform data simultaneously without interference or placing the metrology hardware in a special operating mode.

Per-Application Socket

Each successful subscribe request SHALL result in a socket that is scoped to the requesting application instance. The returned `socket-path` identifies a per-application socket and SHALL NOT be shared across application instances.

Platforms MAY internally multiplex waveform delivery, but the externally exposed socket semantics SHALL be per-application to allow independent access control and isolation.

Alignment Definitions

A waveform stream is *cycle-aligned* when each frame contains an integer number of nominal AC cycles.

A waveform stream is *zero-crossing-aligned* when frame boundaries are aligned to a phase reference, typically the Phase A voltage zero crossing on polyphase devices.

A stream MAY be cycle-aligned without being zero-crossing-aligned. Zero-crossing-aligned implies a phase-reference boundary, while cycle-aligned only describes frame duration.

The `sample-rate-hz` value is the authoritative samples-per-second value. The `samples-per-cycle` value is a nominal descriptor and applications SHALL interpret it together with the stream alignment fields and platform metadata.

15.6.4 Data Format

The waveform data itself is delivered over the returned per-application socket as a native binary frame format. The socket path is runtime state and SHALL NOT be treated as static platform discovery metadata.

`SOCK_SEQPACKET` type sockets provide the application in-order delivery of data that honors message boundaries and provide the sender connection-oriented semantics. Each socket message SHALL contain exactly one complete waveform frame. Partial frames SHALL NOT be transmitted. A data payload frame MAY or MAY not correspond to one or multiple AC cycles depending on the metadata.

Each frame SHALL provide sufficient information for an application to safely consume and interpret the delivered samples without vendor-specific knowledge.

At a minimum, each frame SHALL include:

- A timestamp
- A sequence number
- A payload with multiple sample indexes each with multiple channels

The native socket frame is not a protobuf message. The following C structure is an illustrative native layout for the frame header and typed sample payload:

```
struct geisa_waveform_frame
{
  int64_t    timestamp_ns;    /* first sample time, ns since UNIX epoch␣
↪UTC */
  uint32_t   sequence_num;    /* unspecified starting value */
  uint32_t   reserved;
  union {
    /* variable length array of one sample type */
    int16_t  i16[0];
```

(continues on next page)

(continued from previous page)

```
    int32_t  i32[0];
    float    f32[0];
    double   f64[0];
  } data;
};
```

The timestamp SHALL represent the time associated with the first sample index in the frame and SHALL be expressed as a signed 64-bit integer in nanoseconds since the UNIX epoch (UTC). This defines the timestamp representation and does not imply that the platform clock is accurate to one nanosecond. Timestamp accuracy is platform-dependent.

A monotonically increasing and wrapping sequence number SHALL be included to allow detection of dropped or out-of-order frames. The starting value is unspecified and the sequence number MAY reset on platform restart or when an application re-subscribes. Applications SHALL tolerate resets and gaps.

The sample payload SHALL be encoded using the `sample-type` defined in the waveform metadata.

The number of sample indexes per frame are determined by the socket's read metadata, subtracting the 16 byte header, and dividing by the `total_channel_count` and size of each sample (2, 4, or 8 bytes). For example, if `total_channel_count` is 6, `sample-type` is `int16`, and the socket read returned a frame of 18448 bytes, the frame contains (18448-16)/6/2=1536 indexes.

Samples SHALL be packed in channel-interleaved order per index. For each sample index, all channel values SHALL be present, and channel ordering SHALL be stable and consistent for a given stream.

Within each index, voltage channels are listed first, followed by current channels, using the stable channel ordering described by the stream metadata:

- Time index 0
  - All voltage channels
  - All current channels
- Time index 1
  - All voltage channels
  - All current channels
- Time index 2
  - All voltage channels
  - All current channels
- … continuing for any additional time index in the frame

A standard split phase 2S[1] meter with one voltage and two current channels (three total channels) would report its data as follows:

- Voltage Phase AB, Time 0
- Current Phase A, Time 0
- Current Phase B, Time 0
- Voltage Phase AB, Time 1
- Current Phase A, Time 1
- Current Phase B, Time 1

A standard split phase 12S meter with two voltage and two current channels (four total channels) would report its data as follows:

- Voltage Phase A, Time 0
- Voltage Phase B, Time 0
- Current Phase A, Time 0
- Current Phase B, Time 0
- Voltage Phase A, Time 1
- Voltage Phase B, Time 1
- Current Phase A, Time 1
- Current Phase B, Time 1

For example, a polyphase meter with three voltage and three current channels (six total channels) would report its data as follows:

- Voltage Phase A, Time 0
- Voltage Phase B, Time 0
- Voltage Phase C, Time 0
- Current Phase A, Time 0
- Current Phase B, Time 0
- Current Phase C, Time 0
- Voltage Phase A, Time 1
- Voltage Phase B, Time 1
- Voltage Phase C, Time 1

[1] ANSI defines a series of standard meter "forms" for the North American market in the ANSI C12.1 standard (Code for Electricity Metering).

Each form has specific physical and electrical characteristics.

- Current Phase A, Time 1
- Current Phase B, Time 1
- Current Phase C, Time 1

A polyphase meter with neutral current and three voltage and four current channels (seven total channels) would report its data as follows:

- Voltage Phase A, Time 0
- Voltage Phase B, Time 0
- Voltage Phase C, Time 0
- Current Phase A, Time 0
- Current Phase B, Time 0
- Current Phase C, Time 0
- Current Neutral, Time 0
- Voltage Phase A, Time 1
- Voltage Phase B, Time 1
- Voltage Phase C, Time 1
- Current Phase A, Time 1
- Current Phase B, Time 1
- Current Phase C, Time 1
- Current Neutral, Time 1

All multi-byte integer values SHALL be encoded using host byte order and float values are IEEE 754 encoded.

For integer sample formats (`int16`, `int32`), scaling factors SHALL be applied as defined by the `voltage-scale` and `current-scale` metadata values. For floating-point formats (`float32`, `float64`), scaling factors SHOULD be omitted or set to `1.0`.

The waveform frame format SHALL be defined by this specification and SHALL be sufficient to ensure interoperable decoding across conformant implementations.

The frame structure, including timestamp representation, sequence numbering, sample encoding, channel ordering, and endianness, SHALL be defined such that an application can decode waveform data using only the waveform metadata provided and this specification, without requiring vendor-specific knowledge.

15.6.5 MQTT Details

- QoS: 1 / Acknowledged R/R
- Req Topic: `geisa/api/waveform/req/<userid>`
- Rsp Topic: `geisa/api/waveform/rsp/<userid>`

15.6.6 API Permissions

- Application:
  - Publish: `geisa/api/waveform/req/<userid>`
  - Subscribe: `geisa/api/waveform/rsp/<userid>`

Note

In the topic definitions, `<userid>` refers to the platform-local identifier of the requesting GEISA application instance. See *Glossary* for additional information.

- Platform: - Wildcard Subscribe: `geisa/api/waveform/req/#` - Publish: `geisa/api/waveform/rsp/<userid>`

15.6.7 Transaction Data

- `GeisaWaveform_Req`
- `GeisaWaveform_Rsp`

As defined in https://github.com/geisa/schemas

15.6.8 Examples

Example files for waveform include:

- `waveform-stream-example.json`
- `waveform-subscribe-request-example.json`
- `waveform-subscribe-response-example.json`
- `waveform-unsubscribe-request-example.json`
- `waveform-unsubscribe-response-example.json`

15.7 References

15.8 Actuator Status & Control

15.8.1 Summary

The actuator API allows authorized applications to query actuator state and request actuator state changes.

Actuators are optional and platform/device-specific. Examples include a service switch, DER switch, and load-control or auxiliary relays.

15.8.2 Permission and safety model

Access to actuator status and control is governed by deployment manifest policy and platform policy.

A platform MUST reject unauthorized actuator requests.

A platform MAY reject otherwise well-formed requests if the actuator is unavailable, unsupported, malfunctioning, unsafe to operate, interlocked, administratively disabled, or otherwise denied by policy.

This API defines message structure and request/response behavior. It does not define detailed utility switching workflows, sequencing, tariff behavior, customer notice behavior, or physical interlock behavior.

15.8.3 Actuator targets

This version names the following actuator targets:

- service switch
- DER switch
- load-control relays 0-3

Platforms may expose only a subset of these targets.

Future versions of this specification may add additional actuator types as appropriate, especially as GEISA is expanded to support distribution automation devices.

15.8.4 Getting actuator status

The application publishes `GeisaActuatorGet_Req` on:

`geisa/api/actuator/get/req/<userid>`

The platform responds with `GeisaActuatorGet_Rsp` on:

`geisa/api/actuator/get/rsp/<userid>`

The response includes `GeisaStatus` and `GeisaActuatorStatus` values.

15.8.5 Setting actuator state

The application publishes `GeisaActuatorSet_Req` on:

`geisa/api/actuator/set/req/<userid>`

The request may include one or more desired actuator settings.

The platform responds with `GeisaActuatorSet_Rsp` on:

`geisa/api/actuator/set/rsp/<userid>`

The response status indicates request outcome. A set response does not by itself guarantee completed physical motion; applications should query current status after a request when confirmation of state is required or desired.

15.8.6 MQTT Details

QoS: 1 / Acknowledged R/R

Get:

- request: `geisa/api/actuator/get/req/<userid>`
- response: `geisa/api/actuator/get/rsp/<userid>`

Set:

- request: `geisa/api/actuator/set/req/<userid>`
- response: `geisa/api/actuator/set/rsp/<userid>`

Note

In the topic definitions, `<userid>` refers to the platform-local identifier of the requesting GEISA application instance (see *Glossary* for additional information).

15.8.7 API Permissions

Application permissions:

- Publish `geisa/api/actuator/get/req/<userid>`
- Publish `geisa/api/actuator/set/req/<userid>`
- Subscribe `geisa/api/actuator/get/rsp/<userid>`
- Subscribe `geisa/api/actuator/set/rsp/<userid>`

Platform permissions:

- Wildcard Subscribe `geisa/api/actuator/get/req/#`
- Wildcard Subscribe `geisa/api/actuator/set/req/#`
- Publish `geisa/api/actuator/get/rsp/<userid>`
- Publish `geisa/api/actuator/set/rsp/<userid>`

15.8.8 Transaction Data

Actuator payloads are defined by:

- `GeisaActuatorGet_Req`
- `GeisaActuatorGet_Rsp`
- `GeisaActuatorSet_Req`
- `GeisaActuatorSet_Rsp`
- `GeisaActuatorStatus`

`GeisaActuatorStatus` includes discrete state and optional position fields. Position is meaningful only for actuator types and platforms that support a position or analog setting. Where position is not present, applications should rely on discrete state and response status.

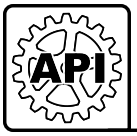
Note that JSON examples and JSON schema are human-readable schema representations, while runtime API payloads are protobuf.

Reference: defined in https://github.com/geisa/schemas

15.8.9 Notes

Actuator visibility and availability are platform specific and deployment-specific.

15.9 Sensors



The GEISA Sensors API provides access to non-streaming, lower-rate sensor data available to edge applications. It supports both request/response for explicit reads and platform-driven publish/subscribe updates.

High-frequency or streaming sensor data is handled separately by the *Waveform* API.

GEISA maintains clear boundaries between APIs while allowing for future sensor types to be integrated as needed. The Sensors API is intended for discrete or periodic measurements, while higher-frequency data is handled by other APIs.

The GEISA platform exposes available sensors through *Platform Discovery*. Applications typically retrieve discovery information once during startup after becoming operational. Discovery information is treated as static for the lifetime of an application instance. If the platform's sensor inventory or sensor capabilities change, the platform restarts applications so they may repeat the discovery process and adapt if needed to the new environment.

For Sensors API topics that include `<userid>`, `<userid>` is a platform-local identifier for the deployed GEISA application instance. It is used for topic routing and local correlation and is not a human user identifier. See *Glossary*.

The Sensors API supports two interaction patterns:

1. request/response, for explicit read or refresh of one or more sensor values
2. publish/subscribe, for platform-driven publication of current sensor observations to authorized subscribers

Reporting cadence for published sensor data is determined by platform and sensor policy, not independently by each subscribing application.

Sensor data access is subject to platform policy, including topic and message-bus access control.

15.9.1 Scope

The Sensors API covers lower-rate, semantically typed sensor observations. It does not include waveform or high-frequency streaming sensor types.

Sensor descriptors are discovery-oriented metadata. Runtime readings are transported separately from descriptors. A generic runtime value model enables support for unknown or future sensors without defining a separate GEISA API for every sensor class.

15.9.2 Sensor Discovery Model

This specification defines GEISA sensor types for interoperability. Platforms shall use a GEISA-defined sensor type whenever an applicable type exists.

The GEISA platform exposes available sensors through *Platform Discovery*.

Each sensor descriptor is represented by `GeisaSensorDescriptor` and includes:

- `sensor_id` (required): platform-local unique identifier for the sensor
- `sensor_type` (required): GEISA-defined interoperable sensor type
- `custom_sensor_type` (conditional): required when `sensor_type` is `GEISA_SENSOR_TYPE_CUSTOM`
- `sensor_subtype` (optional): platform-specific refinement of the sensor type
- `unit` (required): engineering unit used for reported values
- `supports_read` (required): whether explicit read is supported
- `supports_publish` (required): whether platform-driven publication is supported (the platform is capable of pushing sensor data without explicit read requests; updates may be periodic or event-driven; configuration of publishing behavior is implementation-specific as of this version of the specification)
- `min_report_period_ms` (optional): minimum reporting period
- `max_report_period_ms` (optional): maximum reporting period
- `model` (optional): model identifier, useful for troubleshooting
- `name` (optional): platform-defined display name
- `description` (optional): platform-defined description
- `manufacturer` (optional): manufacturer name, useful for troubleshooting
- `geolocation` (optional): geospatial metadata for the sensor or hosting device

The `unit` field defines the engineering unit used for reported values. For sensors that return a single value, it is the expected unit for that value unless overridden in the runtime reading. For sensors that return multiple values in a single reading, `unit` applies only when all returned values share the same unit as of this version of the specification. When multiple values have different meanings or units, the interpretation SHALL be defined by the sensor descriptor, sensor subtype, or platform-specific documentation. This may be revised in a future specification version.

Platforms shall use a GEISA-defined sensor type whenever an applicable type exists. If no applicable GEISA-defined sensor type exists, a platform may use `GEISA_SENSOR_TYPE_CUSTOM` together with `custom_sensor_type`.

Optional platform-specific refinements, such as `sensor_subtype` or other descriptive metadata, does not replace the GEISA sensor type.

The platform may provide identifying information such as manufacturer or model to aid in troubleshooting and debugging.

A platform implementation may also provide implementation-specific descriptive metadata, such as a display name or reporting periodicity, as defined by the corresponding message schema.

This approach preserves interoperability across platforms while still allowing future and platform-specific sensors to be integrated into the system.

15.9.3 Geolocation Metadata

`GeisaGeolocation` is an optional descriptor metadata field for a sensor or hosting device. It contains:

- `latitude`
- `longitude`
- `altitude_m`: optional altitude in meters

This geolocation metadata represents the device or sensor position in the field and is distinct from service point address, customer address, or any PII-associated data.

Geolocation is descriptor metadata only. It is not currently used as a runtime sensor value payload in this version of the specification. Runtime geolocation sensor readings may require a structured value model extension in a future version of the GEISA specification.

15.9.4 Sensor Type Enumeration

The `GeisaSensorType` enumeration defines the following GEISA-defined sensor types for interoperability:

Name	Value	Notes
GEISA_SENSOR_TYPE_UNSPECIFIED	0	Unspecified sensor type
GEISA_SENSOR_TYPE_TEMPERATURE	10	Environmental / physical measurement
GEISA_SENSOR_TYPE_HUMIDITY	20	Environmental / physical measurement
GEISA_SENSOR_TYPE_PRESSURE	30	Environmental / physical measurement
GEISA_SENSOR_TYPE_VOLTAGE	40	Electrical measurement, non-meter
GEISA_SENSOR_TYPE_CURRENT	50	Electrical measurement, non-meter
GEISA_SENSOR_TYPE_POWER	60	Electrical measurement, non-meter
GEISA_SENSOR_TYPE_ENERGY	70	Electrical measurement, non-meter
GEISA_SENSOR_TYPE_CONTACT_STATE	80	State / event-oriented sensor
GEISA_SENSOR_TYPE_TAMPER	90	State / event-oriented sensor
GEISA_SENSOR_TYPE_TILT	100	Motion / physical condition
GEISA_SENSOR_TYPE_VIBRATION	110	Motion / physical condition
GEISA_SENSOR_TYPE_GEOLOCATION	120	May require structured runtime value support
GEISA_SENSOR_TYPE_MICRO_ARC	130	Generic first-pass classification
GEISA_SENSOR_TYPE_SEISMIC	140	Generic first-pass classification
GEISA_SENSOR_TYPE_CUSTOM	500	Use only when no GEISA-defined type applies

15.9.5 Runtime Data Model

Runtime sensor messages carry current or recently observed readings. A runtime sensor reading is represented by `GeisaSensorReading` and includes:

- `sensor_id` (required): sensor identifier, matching a discovered descriptor
- `timestamp_ms` (required): UTC epoch time in milliseconds when the reading was generated
- `values` (required): one or more `GeisaSensorValue` entries for the observation
- `unit` (optional): unit override for the reading
- `quality` (optional): quality indicator
- `status` (optional): status text, such as `OK` or `WARNING`

Most sensors will likely provide a single value, but some sensors may provide multiple related values at the same timestamp. When multiple values are returned, their meaning and ordering SHALL be defined by the sensor descriptor, sensor subtype, or platform-specific documentation until a future GEISA version may define a richer structured model.

For single-value readings, `unit` overrides the descriptor unit. For multi-value readings, `unit` is only appropriate when all values share the same unit. Otherwise, unit interpretation SHALL be defined by the sensor descriptor, sensor subtype, or platform-specific documentation.

The runtime value model is intentionally generic so that unknown or future sensor types can be added without requiring a new GEISA API for each sensor class.

15.9.6 Sensor Value Model

`GeisaSensorValue` is a generic scalar value model. Each value entry contains exactly one of the following fields:

- `double_value`
- `int64_value`
- `bool_value`
- `string_value`

Each individual value remains scalar in this version. A reading may contain one or more values. Future GEISA revisions may introduce richer structured types for self-describing multi-value observations.

Informational mapping guidance:

- `double_value`: temperature, humidity, pressure
- `int64_value`: voltage, current, energy
- `bool_value`: contact-state, tamper
- `string_value`: fallback, vendor-defined, or custom values

15.9.7 Read Request and Response

`GeisaSensorReadings_Req` requests one or more sensor readings. It contains a repeated `sensor_id` field. An empty list MAY be interpreted as a request for all readable sensors accessible to an application.

`GeisaSensorReadings_Rsp` returns zero or more sensor readings. It includes:

- `status` (required): `GeisaStatus` result status
- `readings` (required): zero or more `GeisaSensorReading` entries

Example Sensor Types

This specification does not require a fixed exhaustive set of sensor types. Implementations as well as specific devices may expose sensor types supported by the platform, provided they are represented through the generic sensor contract.

Example platform-local identifiers include:

- `board_temp1` for internal board or electronics temperature
- `ambient_temp1` for ambient or enclosure temperature

For example, a platform may expose:

- board_temp1 with sensor_type set to GEISA_SENSOR_TYPE_TEMPERATURE and engineering unit C
- ambient_temp1 with sensor_type set to GEISA_SENSOR_TYPE_TEMPERATURE and engineering unit C

15.9.8 MQTT Details

- QoS: 0 / Unacknowledged
- Topic: geisa/api/sensor
- Topic: geisa/api/sensor-req/<userid>
- Topic: geisa/api/sensor-rsp/<userid>

Note

In the topic definitions, <userid> refers to the platform-local identifier of the requesting GEISA application instance (see *Glossary* for additional information).

15.9.9 API Permissions

- Application:
  - Subscribe: geisa/api/sensor
  - Publish: geisa/api/sensor-req/<userid>
  - Subscribe: geisa/api/sensor-rsp/<userid>
- Platform:
  - Publish: geisa/api/sensor
  - Subscribe: geisa/api/sensor-req/#
  - Publish: geisa/api/sensor-rsp/<userid>

15.9.10 Transaction Data

- GeisaSensorDescriptor
- GeisaSensorValue
- GeisaSensorReading
- GeisaSensorReadings_Req
- GeisaSensorReadings_Rsp
- GeisaGeolocation

as defined in https://github.com/geisa/schemas

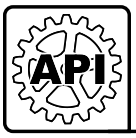
15.9.11 Notes

- Sensor descriptors are exposed through *Discovery* rather than through the Sensor runtime topics.
- Published sensor update frequency is determined by the platform and sensor policy, not independently by each subscribing application.
- Runtime readings contain one or more scalar values. The meaning and ordering of multi-value readings are not yet self-describing in the schema.
- Some sensor types, such as geolocation, may require richer structured runtime value support in future versions of the GEISA specification.

15.9.12 References

- *Discovery*
- *Instantaneous Data*
- *Waveform Data*
- *Platform Status*

15.10 Off-Device Communication



GEISA allows applications to make use of 2 types of off-device communications: Message based, and IP socket based.

GEISA distinguishes between platform-mediated message-based communication and application-managed IP socket communication. These are separate communication mechanisms. Applications do not use GEISA app-message request and response payloads to perform IP socket communication. For ADM-conformant platforms, app-message transport uses the GEISA ADM LwM2M over CoAP path, carried by platform-controlled IP communication outside of the application execution environment. Applications remain unaware of those ADM transport details. Applications that require both mechanisms must request both in their manifest, and platforms enforce each according to the approved deployment policy.

15.10.1 Message-based via LwM2M

This type of communication is available on every device with *Application & Device Management* and uses the administrative LwM2M connection for transport.

This type of communication is bi-directional and messages may be sent unsolicited by either side. Messages can be sent from the application (upstream) and to the application (downstream).

Applications use the message bus to send and receive these messages to/from the platform which in turn makes use of LwM2M to transport them. The applications themselves are unaware of any LwM2M details.

Delivery of messages under some circumstances may be significantly delayed or lost. The *Application & Device Management* is responsible for queuing these messages in both directions.

For upstream messages, the queue contents may be long-lived due to delivery delays or network outages. The messages in the queue SHALL be persistent across device reboots, power cycles, or application restarts, and if undeliverable beyond a timeout period a failure status is reported to the sender.

Note

Message queue persistence, retention, retry behavior, and recovery pacing are platform-managed behaviors subject to operator policy. This version of the specification does not define interoperable configuration mechanisms for these behaviors. Future versions may define standard configuration and observability mechanisms for platform-managed message queues.

Platforms SHOULD avoid creating a burst of delayed traffic when connectivity is restored. Appropriate mechanisms may include message expiration, priority handling, backoff, rate limiting, queue limits, or other platform-controlled delivery controls.

A message-specific time-to-live value, when present, indicates the sender's requested maximum useful lifetime for that message. The platform MAY apply an implementation-defined maximum queued-message retention period and SHOULD expire the message when either the applicable time-to-live or retention period is reached.

Note

This version of the specification does not define an interoperable operator override mechanism for storm conditions, network congestion, FAN outages, or other traffic management events. Such controls may exist as platform-specific or operator-specific capabilities and may be defined in a future version of this specification.

This version of the specification does not mandate fallback or transfer of queued messages across destination classes or communication mechanisms. For example, a platform is not required to transfer queued message-based communication to an application IP socket communication path, or to

transfer failed application IP socket communication to message-based communication. Applications that require cross-mechanism fallback are responsible for implementing that behavior within the permissions approved by the deployment policy.

For downstream messages, the queue contents are usually short-lived if the application is running. If an application is not running, or does not acknowledge message delivery over the message bus, the *Application & Device Management* is still responsible for queuing these messages and, if undeliverable beyond a timeout period, reporting a failure status to the sender.

If an application is uninstalled while its messages are in the queue, a delivery failure status SHALL be sent back upstream for each queued downstream message, and the queue then purged.

Note

Messages that the application sends are deposited into an operator-run system for retrieval by the operator and/or application developer using APIs to the operator's cloud services. The operator or application developer can also initiate messages to the application in bulk or individually.

15.10.2 IP socket based to local devices, private clouds, or public clouds

Applications can also make use of traditional IP socket based real-time communications on devices equipped with cellular, Wi-Fi, or other IP connectivity. Applications MUST specify the endpoints they need to communicate to (IP/Port) in their Application Deployment Manifest so that those policies can be approved by operators and implemented in the platform using firewall rules.

Applications MUST also specify the expected volume of data per day per destination class.

Once a network interface is online, the Application may use outbound-initiated AF_INET/AF_INET6 sockets from within the container environment to reach the specified endpoints.

Approval of IP socket communication does not imply unmanaged or application-selected network reachability. The System Operator remains authoritative for which destination classes, endpoints, protocols, ports, and volumes are allowed. Platform implementations MAY satisfy approved IP socket communication using direct routing, NAT, firewall rules, forwarding, transparent proxies, shared upstream connections, or other platform-controlled mechanisms, provided the Application can use standard socket APIs from within its execution environment and the approved policy is enforced.

Approved IP socket communication may be satisfied through proxying, NAT, forwarding, or shared upstream connectivity. Regardless of the platform mechanism used, the Application remains responsible for its application-layer security behavior. The operator remains responsible for the deployment policy, operator-managed trust configuration, and operation of operator-managed platform and EMS components. See Security considerations below.

Inbound connections towards the Application are only supported for the local destination class

defined below and disallowed from other destination classes.

Note

Application-visible IP socket communication from within the Application execution environment is NOT REQUIRED by every GEISA device. ADM-conformant platforms still require IP connectivity for the ADM LwM2M over CoAP path, but that platform-managed IP connectivity is distinct from Application-managed IP socket access. Applications can be designed to require IP socket communication or, if it is not available, fall back to message-based communication functionality.

Additionally, not all destination classes are REQUIRED by a GEISA device. An operator or device manufacturer may only support a subset depending on capabilities, deployment architecture, or security policy. The availability of an approved application IP communication path is not guaranteed. Applications may encounter intermittent or prolonged outages, or may run on devices where some or all destination classes are not provisioned. Applications approved for IP socket communication SHOULD be designed to handle these conditions, operate in a degraded mode where appropriate, avoid producing excessive alarms or logs, and fall back to message-based communication only when that behavior is implemented by the application and allowed by deployment policy.

15.10.3 Interface Types

The method of connectivity that a device provides can vary from a home Wi-Fi device (e.g. router) providing Internet connectivity, to private Wi-Fi for partner integrations, or to a variety of operator-maintained connectivity options such as shared Wi-Fi hotspots, cellular, mesh, wired, or proprietary interfaces.

Applications need to know which (or multiple) of these are available and online for use at any given time; however, enumerating both the technology and the allowable use cases that they can transport can become complex quickly.

Example types of network interfaces:

- Wi-Fi - Home (device is client)
- Wi-Fi - Meter (device is AP)
- Wi-Fi - Operator/Partner-Hotspot (device is client)
- Cellular - Internet APN
- Cellular - Private APN
- Ethernet - Home
- Ethernet - Private/Partner Devices
- Field Area Network

Other network interfaces that may be available (but out of scope of *IP socket*-based communication in GEISA at this time) are:

- Bluetooth / Bluetooth LE
- IoT integrations via 802.15.4 technologies (Zigbee, Thread, etc.)
- Other wired connectivity or peripherals: RS232/485, USB connected devices, etc.

15.10.4 Destination Classes

The device may have zero, one, or multiple interfaces online at any given time. Applications are not required to know these details but instead only need to know if their desired endpoints are accessible under the approved deployment policy and what volume limits are in place.

Each endpoint an application defines falls into one of these categories:

- Internet Endpoint (ex: Public Cloud)
- Operator Endpoint (ex: Private Cloud)
- Local Endpoint (something off-device, but on-net and private)
- Message based (LwM2M)

For each of these classes, applications have a defined volume limit for when that class is metered.

15.10.5 Network State

For each destination class, applications have a network state that informs the application if and what volume of communication is available as described in *Platform and App Status*.

An application can request communication with more than one class of endpoint and would need a network status indicator for each class separately. For example, it may communicate with both Internet Endpoints as well as Local Endpoints.

Applications may observe a mismatch between reported network state and their own communication attempts. This version of the GEISA specification does not define a dedicated application-to-platform diagnostic reporting mechanism. Implementations MAY expose troubleshooting, logging, or diagnostic reporting through implementation-specific mechanisms or through future GEISA status, logging, or messaging extensions.

15.10.6 Volume Limits

Each destination class may be metered depending on which underlying technology transports the data. A Home Wi-Fi network would normally be considered unlimited whereas a cellular connection would be metered in order to keep the device under a monthly volume limit.

An application developer MUST define volume limits per destination class in their Vendor Application Manifest. These limits may be overridden by the operator at deployment time when converting the Vendor Application Manifest into a Deployment Manifest.

These volume limits are specified as a per day (24 hour period) limit in bytes. For interoperability, application-visible volume accounting is based on application payload bytes. For IP socket communication, this is measured at the application socket boundary. For message-based communication, this is measured at the GEISA app-message payload boundary. Both transmit and receive application payload bytes count toward the application's limit.

Operators may account for expected protocol, security, encapsulation, tunnel, or lower-layer overhead when setting limits. The platform MAY separately measure or estimate interface-level bytes, including protocol and lower-layer overhead, for operator visibility, network planning, analysis, audit, and subsequent adjustment of limits.

The daily rollover mechanism and reset period are deployment and platform policy. This version of the specification does not define an application-visible transaction for configuring those policy values. The application obtains the remaining volume limits and the next reset time via *Platform and App Status*.

If an application exhausts its metered volume quota for one or more destination classes, the policy for those classes remains *metered* and the remaining metered byte count is reported as zero until the next reset or operator adjustment. Exhaustion of a metered quota does not change the policy to *zero*.

The *zero* policy indicates that the Application currently has no usable allocation for that destination class, such as where access is not approved by deployment policy. Volume limits do not apply to destination classes when the policy is *unlimited*.

15.10.7 Security considerations

For message based communications, the operating environment will provide encryption and authentication for data passed between the device and head-end via LwM2M, applications MAY perform further encryption and/or authentication on top of what *Application & Device Management* provides.

For IP socket based communications, the application is responsible for encryption and authentication of data passed between the device and its endpoints where needed.

Applications MUST NOT assume that approved IP socket communication provides the same security, queuing, retry, or store-and-forward behavior as GEISA message-based communication unless those behaviors are explicitly provided by the application protocol or deployment environment.

When a platform uses proxying, forwarding, NAT, tunneling, or shared upstream connectivity, application-layer security remains the responsibility of the relevant communication design. Network-layer mediation can preserve end-to-end application TLS or mTLS semantics. Applications MAY use certificate pinning, client authentication, or other application-layer trust mechanisms where supported by their protocol.

Note

This version of the specification does not define an interoperable mechanism for an Application vendor to declare whether TLS inspection, explicit proxying, or operator-mediated re-origination of application-layer security is supported. Operators and platform implementations that use such

mechanisms are responsible for ensuring that the behavior is compatible with the Application, its configured trust material, and the deployment policy. A future version may define manifest fields or policy controls for this behavior.

15.10.8 Connectivity

The operating environment is responsible for providing network connectivity between each Application container environment and network interfaces.

The platform is responsible for implementing policy (by firewall and forwarding rules), accounting (volume limits), and providing underlying connectivity and routing between these components.

Note

GEISA does not mandate a specific technology that the implementer of the operating environment must use to accomplish this, but does recommend the use of Linux network namespaces, veth interfaces, and iptables/nftables for filtering, NAT, and accounting. The implementer may also make use of on-device transparent proxies if desired, however the Application must be able to use AF_INET/AF_INET6 sockets from within the container environment with any encoding and protocol within.

The operating environment MUST provide a local lo interface within the container environment for each application. The local lo interface must be up and configured with both 127.0.0.1 and ::1 addresses.

The apparent address, route, or connection path visible to an Application inside its container environment is not required to correspond one-to-one with the physical interface, upstream session, or operator network path used by the platform. Applications should treat the socket interface as the approved local execution-environment abstraction and rely on Platform and App Status for communication availability and quota state.

15.10.9 Policy Rules

The *Application Manifests* include a set of Endpoints the Application is expecting to communicate with. The application must list every endpoint that it wishes to communicate with per destination class.

15.10.10 DNS

Warning

TODO: should we forgo DNS for GEISA 1.0?

While policy rules are defined using IP addresses, Applications may use DNS queries to avoid hard-coded IP literals within their application source or configuration files.

The Operating Environment and Network Manager must provide DNS services for applications to use, however the scope of resolution may be limited particularly for devices that are not or are poorly Internet connected.

GEISA highly recommends that the Operating Environment implement a caching local resolver that honors TTL to reduce network traffic off-device due to repeated application lookups for the same name.

The application must be able to reach DNS services from within the container environment by using standard Linux libraries (i.e., libnss/resolvconf/etc.)

DNS in a multi-tenant and multi-interface environment can get quite complex. For example, an operator may implement their Operator Endpoints using a dedicated private TLD and configure the resolver to direct DNS lookups for that TLD over their private network where other TLDs for Internet Endpoints use a home Wi-Fi or Internet-connected Cellular provider's supplied DNS.

15.10.11 Local Endpoint Considerations

Local Endpoints are defined to allow an Application access to local resources on a connected network. Typically this would be for purposes of local device integrations such as EVSE, Solar Inverter and battery storage equipment, smart plugs and similar devices. This class is typically only available for devices that are connected to a home or Meter hosted Wi-Fi, or wired connection, for example.

Unlike the operator and Internet destination classes, the local destination class allows additional functionality:

- Inbound access, including multicast and broadcast
- Outbound multicast and broadcast

As the IP addressing of the connected network are not known and not static between devices, the Application Manifest cannot list destination/source IP addresses for policy rules, but instead only protocol/port and multicast groups.

An application requesting inbound access in its manifest requires that the platform create that inbound mapping from the appropriate local interface (such as a WiFi interface) into the application container environment. When requesting a port mapping, both IPv4 and IPv6 MUST be mapped if the local interface supports both.

Many common local device protocols use multicast or broadcast for discovery and registration. Applications SHOULD be able to both send to IP multicast groups and register IP multicast groups for receipt. If specified in the application manifest, the platform will register those addresses on the

appropriate local interface and forward received packets that match both the multicast/broadcast address and the inbound protocol/port into the application container environment.

Multiple applications MAY register the same IP multicast group at the same time; however, they MUST NOT register the same protocol/port. The platform and/or ADM system MUST arbitrate inbound protocol/port requests in each application manifest to prevent multiple applications from requesting the same inbound protocol/port combination.

MQTT Details

- QoS: 1 / Acknowledged R/R
- Req Topic: `geisa/api/message/upstream/req/<userid>`
- Rsp Topic: `geisa/api/message/upstream/rsp/<userid>`
- Req Topic: `geisa/api/message/downstream/req/<userid>`
- Rsp Topic: `geisa/api/message/downstream/rsp/<userid>`

Note

In the topic definitions, `<userid>` refers to the platform-local identifier of the requesting GEISA application instance (see *Glossary* for additional information).

API Permissions

- Application:
  - Publish: `geisa/api/message/upstream/req/<userid>`
  - Publish: `geisa/api/message/downstream/rsp/<userid>`
  - Subscribe: `geisa/api/message/upstream/rsp/<userid>`
  - Subscribe: `geisa/api/message/downstream/req/<userid>`
- Platform:
  - Wildcard Subscribe: `geisa/api/message/upstream/req/#`
  - Wildcard Subscribe: `geisa/api/message/downstream/rsp/#`
  - Publish: `geisa/api/message/upstream/rsp/<userid>`
  - Publish: `geisa/api/message/downstream/req/<userid>`

Transaction Data

Warning

Need to add refererence to content within https://github.com/geisa/schemas here.

16 Security

Security is fundamental to the GEISA specification, so much so that it is included in the name itself: Grid Edge Interoperability and **Security** Alliance. This specification was developed with prevailing security and privacy design principles in mind, such as NIST 800-53, CCPA and NERC CIP standards, and enables implementers to incorporate security controls appropriate to their operational contexts and needs.

Security in GEISA exists to enable trusted interoperability across independently developed hardware, software, and operational domains without requiring a single vendor-controlled platform. It is designed to allow applications from different vendors to co-exist, to allow utilities to trust in portable secure workloads and/or edge applications, and allows for interoperability without shared implementations, while allowing the Utility or Operator to dictate their specific policies and standards.

GEISA is intended to support multi-vendor application ecosystems across heterogeneous hardware platforms, and is designed to support deployment across utilities and operators with differing regulatory obligations, risk tolerances, and operational practices. Accordingly, the specification separates capability definition from policy enforcement.

The Common Operating Environment (COE) provides mechanisms that enable security, privacy, and operational policies to be configured by the deploying utility or operator. GEISA does not mandate uniform policy settings, nor should implementations hard-code assumptions about allowable access, connectivity, or application behavior.

The COE is not a single software component or runtime environment. It is the logical collection of execution environments, system services, control interfaces, policy enforcement mechanisms, and lifecycle management functions defined by GEISA that together provide a governed platform for applications at the grid edge.

The COE may include one or more execution environments (e.g. native Linux or virtual runtimes), but is defined by the security, isolation, authorization, and operational controls it enforces rather than by any specific implementation technology.

This approach ensures that:

- Utilities and operators retain authority over what functionality is enabled.
- Policy decisions may be adapted to local regulatory or operational needs as required.
- Interoperability across vendors (hardware and software as applicable) is maintained without constraining governance.

Implementations shall expose the configuration and control interfaces necessary for operator-defined policy to govern behavior within the COE.

The following are the minimum criteria needed for success of the GEISA specification and for the implementor.

- Clear expectations must be illustrated outlining what responsibilities fall to the GEISA team vs the entities or individuals implementing the specification. This will be referred to as the "Shared Responsibility Model".
- The specification must enable implementers to comply with applicable standards, frameworks, certifications, or regulatory requirements.

16.1 Responsibilities

16.1.1 Shared Responsibility Model

GEISA cannot feasibly account for every implementer's operational context, architecture, governance model, policy requirement, or business process. Accordingly, responsibilities are divided between GEISA and those entities implementing or operating solutions based on the specification.

The GEISA specification is architected according to established security design principles; however, as a specification it defines required capabilities and interfaces rather than prescribing specific implementations. It provides the mechanisms and flexibility needed for implementers and operators to apply their own policies, controls, and technology choices appropriate to their environment.

GEISA is therefore not responsible for the final design, implementation, or operation of security controls within a specific implementation or deployment. The specification is not tailored to implementation-specific requirements and does not mandate particular components, libraries, or platforms. It is intentionally implementation-agnostic so that conformant solutions may adapt to evolving technologies, regulatory landscapes, and operational priorities while maintaining interoperability.

GEISA also recognizes that Implementers and Operators adopting the specification may have varying definitions of data classification and policy statements surrounding those classifications. The specification is designed to allow for the application of policy and controls appropriate to a variety of needs and requirements. As such, it and does not mandate specific classifications or associated controls. Instead, GEISA makes capabilities available to Implementers and Operators such that they may apply their own classifications and controls as needed to meet their requirements.

16.1.2 GEISA Responsibilities

- Making available to an implementer a baseline specification for a Common Operating Environment(COE) with levers needed to provide the implementer the ability to meet their applicable security and privacy requirements.
- Identifying the minimum system requirements in order to run the GEISA specification and testing against those system requirements to prove out a reference implementation in support of implementers.
- Implementing processes to ensure that contributions to the GEISA specification are properly vetted and carried out by individuals free of malicious intent.
- Defining and maintaining a baseline *Threat Model* to identify risks, and where deemed appropriate, aligning system design to either mitigate those risks or allow for mitigation of those risks by an implementer using mechanisms available within the COE.

16.1.3 Implementer Responsibilities

You as the implementer are best suited and equipped to understand the architecture and environment that you are implementing within, the parties involved, additional/specific policy requirements and risk appetites, etc. and, as such, the following responsibilities have been identified:

Responsible for:

- Identifying the applicable laws, regulations, required security and privacy controls or other security and privacy-oriented requirements that individual specific entities must satisfy, assessing the GEISA specification for suitability to meet those requirements, and then implementing those requirements.
- Identifying the underlying hardware, choosing a compatible operating system they wish to use and at their discretion any additional software or system services deemed necessary beyond what is defined in the GEISA specification.
- Once implemented, keeping the operating system and any other software current with updates as they are released to fix published bugs, defects, vulnerabilities, etc.
- Defining their own shared responsibility model with any third parties that they may allow to access or provide components to the architecture they implement inclusive of the GEISA specification. This may include but is not limited to first or third-party application developers, system operators, etc.
- Defining and maintaining a threat model, which may take into account the GEISA baseline threat model, and identifying applicable risks as well as any appropriate action(s) required in order to mitigate those risks.
- Ensuring achievement of any/all implementation-specific control requirements spanning any/all relevant Security and Privacy domains.
- The GEISA specification may assist the implementer with achieving certain baseline implementation requirements but it is not responsible for the final design, implementation and/or

operation of controls defined by the implementer nor is the GEISA specification required to meet implementation-specific requirements.

The following threat model establishes the risk framework used to derive the security capability expectations defined by this specification.

16.2 Threat Model

The GEISA threat model identifies the assets requiring protection, the trust boundaries across which the GEISA Common Operating Environment (COE) operates, and representative threat scenarios that conformant implementations should be designed to mitigate. It is not exhaustive and is intended to be a living framework that evolves alongside the threat landscape. It provides implementers with a baseline set of threats to consider when evaluating their specific implementations.

Implementers SHOULD extend this model to reflect their operational context, inclusive of, but not limited to, their deployment, regulatory, and operational environments and identify additional threats applicable to their specific circumstances.

Unlike traditional IT systems, compromise of grid-edge devices and resources may create direct reliability impacts to the electrical grid or to individual service locations. Therefore, prioritizing a defense-in-depth approach is essential. Additionally, the potential for physical access to grid-edge devices creates a need for robust physical security controls and/or tamper resistance.

The GEISA specification is designed to enable implementers to meet these needs through the design of the COE and the control levers it provides to enforce deployment-specific security policy.

16.2.1 Software Provenance and Deployment Authorization

GEISA deployments rely on verifiable software provenance and explicit operator authorization. Trust decisions may involve multiple independent signing or credential authorities, including application providers, utilities/operators, and, where implemented, device or platform providers.

The following roles are referenced in this section:

- *Application Publisher*: the entity responsible for producing, signing, and asserting provenance of an application or workload artifact.
- *Device/Platform Provider*: the entity responsible for the hardware or platform on which the COE executes, when such identity is represented.

Note

Review whether the PKI governance language below reflects current consensus regarding operator authority and interoperability. In particular, hw impact and compute needed for the more frequent validation checks.

GEISA does not mandate a specific PKI hierarchy; however, cryptographic identity, signing, and validation are REQUIRED capabilities.

Implementations shall be capable of interoperating with operator-selected utility, vendor, or industry PKI systems. Conformant implementations shall not require exclusive use of a proprietary trust infrastructure controlled solely by the implementer.

The utility or operator retains final authority over trust anchors, certificate hierarchies, and approval of credential authorities.

Conformant implementations shall support operator-selected trust anchors and credential authorities and shall not constrain the operator to a single implementer-controlled trust infrastructure.

Support for operator-selected trust configurations shall be provided in a manner that preserves the verification and enforcement capabilities described herein.

Conformant implementations must provide mechanisms enabling:

- Validation of application provenance (e.g., verification that an application artifact corresponds to the bits produced by an identified vendor or publisher).
- Validation of operator authorization prior to deployment or activation (e.g., the utility/operator explicitly approves whether a given application is permitted in a target environment, device class, or operational mode).
- Verification of authenticity and authorization in both:
  - application lifecycle workflows (e.g., package intake, cataloging, approval, deployment orchestration), and
  - on-device enforcement prior to install, activation, or execution within the COE.
- Support for both centralized deployment orchestration and authorized local provisioning workflows (e.g., field tool installation by authorized personnel).

Device or platform identity mechanisms (e.g., hardware vendor certificates or attestation) may be used to strengthen trust in the executing environment. GEISA does not mandate such mechanisms; however, if implemented they must integrate with the above validation and enforcement model.

Validation performed by centralized or upstream lifecycle management systems shall not be considered sufficient on its own. The COE must enforce verification of software authenticity and authorization locally on the device prior to installation, activation, or execution. This ensures that software running within the COE is trustworthy even in the event of compromised upstream systems or supply chain attacks that may attempt to introduce unauthorized or malicious software.

The COE is the final technical enforcement point on the device itself. Operational authority remains with the utility/operator (or delegated operational systems) that define policy and authorization; the COE ensures those decisions are enforced locally and cannot be bypassed by external systems. External lifecycle or orchestration systems may request deployment actions, but shall not be able to bypass local verification, authorization, or policy enforcement.

16.2.2 Lifecycle Trust Enforcement

Trust within GEISA is not established at a single point in time. It is evaluated and enforced across multiple lifecycle transitions, including application ingest, approval, deployment, installation, activation, and execution.

Conformant implementations must ensure that trust assertions made by one system or phase are independently verified at the point where enforcement occurs. Trust shall not be implicitly inherited across lifecycle boundaries without validation.

This includes, but is not limited to:

- Verification of application integrity and publisher identity when software is introduced into lifecycle management systems.
- Confirmation of operator authorization prior to deployment to a device or environment.
- Independent validation performed by the COE prior to installation and at activation (including restarts of applications and/or workloads).

> **Note**
>
> Need to discuss execution/runtime or periodic validations.

- Continued enforcement of policy, identity, and behavioral constraints during runtime operation.
- Runtime enforcement of operator-defined behavioral constraints, including the ability to detect, limit, or terminate workloads that violate policy or exceed authorized resource or communication profiles.
- Revocation of trust relationships when software is no longer authorized for execution within the COE (e.g. due to compromise, policy change, or end of life).

This model ensures that compromise of any single lifecycle component does not result in implicit trust of software or actions elsewhere in the system and helps to mitigate risks resulting in systemic compromise.

16.2.3 Trust Revocation and Credential Lifecycle

Note

definitely needs review:

GEISA defines mechanisms (revocation, update, policy change), but the operator or implementer is responsible for monitoring, initiating, and governing these actions. GEISA-compliant implementations and deployments must support the ability to modify or withdraw previously established trust relationships throughout the operational lifecycle of a device or application.

Conformant implementations shall provide mechanisms enabling:

- Revocation or invalidation of application, publisher, operator, or device credentials when trust can no longer be assured.
- Enforcement of operator-directed actions to disable, restrict, or remove software that is no longer authorized for execution within the COE.
- Update and rotation of trust anchors, certificates, and associated cryptographic material without requiring replacement of deployed devices.
- Local enforcement of revocation or policy changes even in environments with limited or intermittent connectivity to centralized management systems.

These capabilities ensure that trust established during provisioning or deployment can be continuously evaluated and adjusted in response to evolving security conditions, operational decisions, or cryptographic requirements.

16.2.4 Assets Requiring Protection

The GEISA specification is designed to protect the following asset classes:

- Integrity of grid-impacting control functions (e.g., service disconnect, DER coordination, or other local controls)
- Authenticity and integrity of telemetry, metrology, and status data
- Device identity and associated cryptographic credentials
- Availability of edge execution environments necessary for safe grid operation
- Application and Application Publisher identity as represented by associated credentials
- Isolation between workloads executing within the COE
- Application and device monitoring, behavioral logging, and alerting capabilities
- Customer data subject to privacy and/or regulatory protection
- Trust relationships with utility/operator systems
- Software and firmware integrity across the device lifecycle

16.2.5 Trust Boundaries and External Interfaces

The GEISA Execution Environment operates across multiple security trust boundaries. A trust boundary exists wherever administrative control, execution privilege, or assurance level changes.

GEISA defines these boundaries to enable secure interoperability among heterogeneous vendor solutions while preserving the operator's ability to enforce deployment-specific policy. The GEISA Common Operating Environment (COE) acts as the on-device enforcement point through which operators may permit, restrict, or condition interactions between applications, networks, and connected systems.

These enforcement functions exist regardless of whether applications execute locally, participate in hybrid edge/cloud workflows, or are installed through centralized lifecycle management systems.

All interactions crossing these boundaries shall be treated as untrusted unless explicitly authenticated, authorized, and governed by operator-defined policy. The boundaries listed below represent the primary attack surfaces considered by this model. They are not exhaustive. Implementers shall evaluate their deployment to determine whether additional boundaries are present.

Note that examples are illustrative and do not define required sensor types or configurations.

Boundary	Description	Examples
Field Devices	Physically accessible deployment environment where attackers may gain hardware access. This includes attempts to tamper with the device, extract sensitive material, or conduct side-channel to observe, influence, or bypass normal system protections.	Enclosure access, debug ports
Local Provisioning / Maintenance	Interfaces used during installation or maintenance that may introduce temporary elevated trust.	Field tools, Bluetooth, serial
Sensors	Data originating from instrumentation whose acquisition and integrity are governed by the COE device itself.	Metrology, temperature, accelerometer, GNSS, other directly-attached sensors
Customer Networks	Communications over networks not managed by the utility/operator and therefore not inherently trusted.	Home/Business Wi-Fi, HAN, LAN, Thread
Workload (e.g. Applications)	Separation between the COE platform and hosted workloads enabling operator control over resource and network access. This boundary exists even when all code is local due to multi-tenancy, privilege separation, and differing lifecycles.	GEISA COE APIs, resource allocation, storage, permission models
External Device / Control	Interfaces to autonomous systems not governed by GEISA lifecycle or assurance controls. Includes third-party devices or subsystems that may have grid or local impact if compromised.	Inverters, Matter devices, thermostats, gateways, controllable loads, local data sources
Operational Integration	Exchange of operational data with upstream systems, including utility or non-utility infrastructure outside the COE trust domain.	AMI backhaul, DERMS, analytics platforms, monitoring systems, hybrid edge/cloud applications
Lifecycle Management	Authority to deploy, configure, approve, restrict, update, or revoke software and policy governing the COE.	Application deployment, update orchestration, policy/config distribution

Utility-operated networks and systems are considered external to the COE trust domain and are there-

fore modeled within the Operational Integration or Lifecycle Management boundaries depending on function.

Devices communicating via customer-managed ecosystems (e.g., Wi-Fi, Thread, Matter, or similar technologies) are not considered part of the Sensor Boundary, even when providing measurement data. Such devices operate with independent identity, firmware, and lifecycle control and are therefore treated as External Devices whose communications traverse the Customer Network Boundary.

16.2.6 Threat Actors Considered

The GEISA threat model considers a range of potential threat actors with varying levels of access, capability, and intent. These actors may operate independently or in coordination and may target individual devices, aggregated infrastructure, or upstream systems.

The following categories are considered within scope:

Threat Actor	Description
Supply Chain Adversary	Actor seeking to introduce malicious components, firmware, or updates during manufacturing or distribution stages.
Field-Level Physical Attacker	Actor with physical access capable of tampering, extraction, or hardware manipulation to disrupt operation, alter executable code outside authorized lifecycle processes, or obtain data from the device. This may include invasive techniques (e.g., device/chip decapsulation (“decap”), probing, or unauthorized reflashing) intended to bypass protections.
Opportunistic Local Network Attacker	Actor on a customer or shared network capable of interception, manipulation, or disruption of communications or operations.
Unauthorized Local Service Actor	Individual attempting access via provisioning or maintenance interfaces using unauthorized or stolen tools or credentials.
Remote Adversary	External attacker using network access to exploit exposed services, protocols, or integrations.
Coordinated Distributed Actor	Group or automated campaign capable of influencing behavior across many endpoints.
Compromised Upstream System	Trusted operational or management system issuing valid-looking but malicious actions after being compromised.
Compromised External Device	Third-party or customer-managed device providing malicious data or attempting lateral influence on the COE.
Malicious or Negligent Insider	Individual with authorized access who may misuse privileges, attempt escalation, alter policy, or exfiltrate information, either intentionally or unintentionally.

16.2.7 Representative Threat Scenarios

The following scenarios illustrate representative ways in which identified threat actors may attempt to exploit trust boundaries to impact protected assets. These examples are not exhaustive but demonstrate the types of risks implementations should be designed to address.

Implementations SHOULD address these risks using deployment-specific controls, policies, and technologies consistent with the capabilities defined by the GEISA specification.

16.2.8 Unauthorized Use of Local Service Interfaces

An Unauthorized Local Service Actor connects to provisioning or maintenance interfaces using stolen credentials or unauthorized tools to modify configuration, deploy software, or alter policy.

- Boundaries Involved: Local Provisioning / Maintenance, Lifecycle Management
- Potential Impact: Unauthorized deployment, policy manipulation, integrity loss, or unauthorized access to sensitive configuration or data.

16.2.9 Compromise of a Customer-Network Connected Device

A Compromised External Device (e.g., Matter or Wi-Fi system) provides falsified telemetry or malformed data intended to influence processing performed locally or by upstream systems participating in hybrid workflows.

- Boundaries Involved: External Device / Control, Customer Networks
- Potential Impact: Corrupted data, unintended control actions, incorrect operational conclusions, or propagation of misleading information into coordinated services.

16.2.10 Malicious or Altered Software Introduced During Update

A Supply Chain Adversary inserts malicious code into firmware, applications, or update packages prior to deployment.

- Boundaries Involved: Lifecycle Management
- Potential Impact: Persistent compromise, unauthorized control, data exposure, or loss of software and device integrity across the lifecycle.

16.2.11 Remote Exploitation of Exposed Services

A Remote Adversary targets reachable services or protocols to gain unauthorized access, manipulate data, or disrupt availability.

- Boundaries Involved: Operational Integration
- Potential Impact: Loss of availability, unauthorized command execution, unintended control actions, or compromise of data privacy.

16.2.12 Insider Abuse of Authorized Privileges

A Malicious or Negligent Insider misuses authorized access to escalate privileges, alter logging or policy, access sensitive data, or deploy unauthorized workloads.

- Boundaries Involved: Workload Boundary, Lifecycle Management
- Potential Impact: Policy bypass, unauthorized data access, compromise of data privacy, undetected system changes, or loss of operational integrity.

16.2.13 Coordinated Manipulation Across Multiple Devices

A Coordinated Distributed Actor attempts to influence many endpoints simultaneously to create aggregate operational impact or to collect or manipulate data at scale.

- Boundaries Involved: Operational Integration
- Potential Impact: Grid instability, synchronized disruption, large-scale data exposure, or propagation of misleading information into upstream systems.

The following capability expectations describe the security functions that implementations must be able to realize in order to address the risks identified in this threat model.

16.3 Interpretation of Conformance

The threat model is informative and provides context for required capabilities; it does not prescribe specific mitigations or implementation techniques.

GEISA conformance is evaluated based on the ability of an implementation to demonstrate the security capabilities and enforcement behaviors defined by this specification, rather than on the use of any specific technology, framework, or software component.

Conformance testing may exercise defined interfaces, workflows, and operational scenarios to validate that required security properties—such as identity verification, authorization enforcement, workload isolation, integrity protection, and policy governance—are correctly realized.

Implementations may use differing operating systems, execution environments, cryptographic libraries, or architectural approaches, provided that the required behaviors and outcomes are achieved and are externally verifiable through defined interactions.

Where minimum versions or technical baselines are referenced, they shall relate to interoperable standards, protocols, or security capabilities and shall not mandate specific vendor products, libraries, or implementation frameworks.

16.4 Security Capability Expectations

To address the risks identified in this threat model, conformant implementations must realize these capabilities within the COE, which serves as the enforcement layer for application behavior, resource governance, and authorized interaction with connected systems while ensuring that operators retain control of the ability to set and enforce policy, protect system integrity, and maintain trust relationships across defined trust boundaries.

The GEISA specification defines the security outcomes that must be achievable, but does not prescribe specific technologies or implementation approaches.

16.4.1 Identity and Authentication

Implementations shall support verifiable identity for:

- Devices or platforms (e.g., meters, NICs, or other edge hardware), hereafter referred to as the *Device/Platform Provider* where such identity is supplied by the hardware or platform manufacturer.
- Applications and associated software artifacts produced by an *Application Publisher* (the entity responsible for creating and signing an application or workload).
- External systems interacting across trust boundaries.

These identities establish provenance, accountability, and authorization context for software deployment and operation within the COE.

Such identities may represent organizational provenance used to establish trust in software origin and authorization for deployment within the COE.

Identity mechanisms must protect associated credential material from unauthorized extraction, duplication, or misuse and support secure lifecycle management of those credentials.

16.4.2 Authorization and Policy Enforcement

The COE must enable operator-defined policy controlling application execution, resource access, and permitted interactions across networks and systems.

16.4.3 Integrity Protection

Implementations must protect software, configuration, and operational data from unauthorized modification.

16.4.4 Isolation of Workloads

The COE shall provide isolation sufficient to prevent privilege escalation or cross-application interference.

16.4.5 Secure Lifecycle Management

Implementations must support secure deployment, update, validation, and removal of software throughout the device lifecycle.

16.4.6 Protection of Data

Implementations shall protect sensitive data from unauthorized disclosure or manipulation when stored or transmitted across trust boundaries.

16.4.7 Auditability and Visibility

The COE shall enable logging and monitoring sufficient to detect unauthorized activity and support operational accountability.

16.4.8 Resilience and Availability

Security mechanisms shall support continued safe operation and policy enforcement under degraded connectivity conditions.

These capabilities are intended to enable interoperable yet independently governed deployments across diverse utility environments.

17 Revision History

Version 0.1.8 - 2026-01-05 - Integrated VEE submission.

Version 0.1.7 - 2025-12-16 - Reorganized App Isolation, local network access, and updated VEE details.

Version 0.1.6 - 2025-10-20 - Updated ADM section with Wiki materials.

Version 0.1.5 - 2025-09-04 - Reorganized and expanded ADM section.

Version 0.1.4 - 2025-08-04 - Added contributions on security and API. Updated VEE Icons. Reorganized EE sections.

Version 0.1.3 - 2025-07-14 - Updated document to reflect different types of EE conformance.

Version 0.1.2 - 2025-06-30 - Updated build system to automatically build PDF. Updated Waveform API.

Version 0.1.1 - 2025-06-22 - Updated draft with clear ADM, API, + EE headings

Version 0.1.0 - 2025-06-17 - Early (and incomplete) Draft

Bibliography

[LWM2M] OMA Lightweight M2M

[LWM2M-Core] OMA SpecWorks Lightweight Machine to Machine Technical Specification Core 1.2

[LWM2M-Transport] OMA SpecWorks Lightweight Machine to Machine Technical Specification: Transport Bindings 1.2

[RFC2119] Key words for use in RFCs to Indicate Requirement Levels

[RFC7252] The Constrained Application Protocol (CoAP)

Index

U

V