

OPEN
Compute Project

Secure Firmware Recovery

Version 1.1

EDITOR(s): Eric Spada, Broadcom
Varun Sampath, NVIDIA
Mariusz Oriol, NVIDIA

CONTRIBUTORS:

Wojtek Powiertowski, Meta
Ben Stoltz, Google
Bryan Kelly, Microsoft
Vladimir Dreizin, Broadcom
Edmund Szeto, Broadcom
Yigal Edery, NVIDIA
Danny Ybarra, Western Digital
Abishek Anuroop, NVIDIA
Ganesh Sudhir Mirajkar, NVIDIA

Revision History

Revision	Date	Guiding Contributor(s)	Description
0.9	04-13-21	Eric Spada, Broadcom	Draft Release
1.0-rc	06-14-22	Eric Spada, Broadcom	Review Release
1.0	09-14-22	Eric Spada, Broadcom	Final Release
1.01	11-17-22	Eric Spada, Broadcom	INDIRECT_DATA Max data returned is 252B
1.1-rc8	05-22-25	Varun Sampath, NVIDIA	Added flashless boot mode Added multi-stage activation Added I3C support Added FIFO support
1.1	12-19-25	Mariusz Oriol, NVIDIA	Added USB support Clarified Maximum Response Time and Heartbeat Period definition. Updated document formatting

Executive Summary

The environment. Cloud data centers contain servers full of myriad Devices running firmware.

The problem to be solved. Before OCP Secure Firmware Recovery was created, servers had no standardized, open protocol for recovering Device firmware from accidental or malicious corruption.

The Open Compute Project solution. OCP Secure Firmware Recovery defines a protocol for recovering a Device firmware using the SMBus, I3C and USB protocols.

Table of Contents

Executive Summary	3
1. PURPOSE	6
2. AUDIENCE	6
3. SYNTAX AND CONVENTIONS	6
4. INTRODUCTION	6
4.1 RELATIONSHIP TO OTHER OCP SECURITY DOCUMENTS	7
4.2 DEVICE RECOVERY AND SP800-193 ALIGNMENT	7
4.3 DETECTION, REMEDIATION, CORRUPTION AND INITIATING RECOVERY ACTION	7
4.3.1 Types of Failures	8
4.3.2 Response to firmware/configuration failures	8
4.3.3 Administrative Forced Recovery	8
4.4 RECOVERY REQUIREMENTS ACROSS PLATFORM COMPONENTS	9
4.5 RECOVERY AUTHORIZATION	9
4.6 REASONS FOR DEVICE RECOVERY	10
4.7 DEVICE RECOVERY USE CASES	10
4.7.1 Device Software Update	10
4.7.2 Device Recovery with Critical Data	11
4.7.3 Device Recovery without Critical Data	11
5. RECOVERY COMPONENTS AND ROLES	11
5.1 RA & PA-RoT Recovery Role	12
5.2 AC-RoT Recovery Role	13
5.3 RECOVERY IMAGE	14
6. RECOVERY PROCESS	15
7. RECOVERY INTERFACE FUNCTIONS	17
7.1 Device Reset	17
7.2 Forced Recovery	17
7.3 Flashless Boot	17
7.4 Recovery Image Push	17
7.5 Recovery Image Selection	18
7.6 Recovery Image Activation	18
7.7 Recovery Image Authentication and Operation	19
7.8 Normal/Healthy Operation	19
8. RECOVERY INTERFACE	19
8.1 Capability/Discovery	20
8.2 Indirect Memory Interface	20
8.2.1 Addressing within a Component Memory Spaces	20
8.2.2 Code CMS	22
8.2.3 Critical Logging CMS	22
8.2.4 Vendor Defined CMS	22

8.2.5 Indirect FIFO CMS	23
8.3 Recovery SMBus Interface	24
8.3.1 SMBus Topology	25
8.3.2 SMBus Device Addressing and Commands	26
8.3.3 Interface Sharing/Isolation	26
8.4 Recovery I3C Interface	26
8.4.1 I3C Interface implementation notes	28
8.5 Recovery USB Interface	28
8.5.1 USB EP0 Command Encapsulation	30
8.5.2 USB Interface Descriptor	31
8.5.3 USB Interface Functional Descriptor	32
8.5.4 USB Descriptors for OCP Secure Firmware Recovery Interface	33
8.5.5 USB-specific Error Recovery	33
8.5.6 USB Device implementation notes	34
9. RECOVERY INTERFACE COMMANDS	34
9.1 Error Handling/Unsupported Features	34
9.2 Command Summary	35
10. PROTOCOL CONFORMANCE CHECKLIST/STATEMENT	46
11. GLOSSARY AND ABBREVIATIONS	49
12. RELEVANT STANDARDS, GUIDELINES, AND DOCUMENTS	49
13. LICENSE	49
14. ABOUT OPEN COMPUTE FOUNDATION	50

1. Purpose

This document creates guidelines on how to recover a failed or compromised Device. The recovery operation provides a mechanism for a recovery agent (RA), in coordination with a PA-RoT (Platform Active RoT), to recover a Device's firmware and/or security critical parameters of an AC-RoT (Active Component RoT). The recovery process **MUST** bring the Device to a known security state.

2. Audience

The audience for this document includes, but is not limited to, system and system component designers, security information and event management (SIEM) system developers, and cloud service providers.

3. Syntax and conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [BCP 14 \[RFC2119\]](#) [\[RFC8174\]](#) when, and only when, they appear in all capitals, as shown here.

The roles “attester”, “verifier”, and “reference integrity measurements” are defined in the [OCP Attestation of System Components v1.0 Requirements and Recommendations](#)

4. Introduction

Guiding principles for this document are based on [NIST SP 800-193](#) and the three pillars supporting Platform Resiliency:

- Protection – Secure boot/Attestation/Threat Model
- Detection – [Attestation of System Components v1.0 Requirements and Recommendations](#)
- Recovery – This document

This document focuses on the Recovery principle, which is a mechanism for restoring Platform Firmware code and critical data to a state of integrity in the event that firmware code or critical data have been corrupted, the Device is unresponsive, or when forced to recover through an authorized mechanism. While focusing on Recovery, the document will also discuss aspects of Detection and Protection where needed.

The agents involved in the recovery process

- Device or AC-RoT: Device which is being recovered

Date: Dec 19, 2025

This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](#).

- PA-RoT: System component responsible for determining the health of a Device and initiating recovery via the recovery agent.
- Recovery Agent (RA): System component responsible for performing the recovery process.

4.1 Relationship to Other OCP Security Documents

The threat model is described in the [OCP Common Security Threats v1.0](#). When attestation is referred to in this document, it assumes compliance with the [OCP Attestation of System Components v1.0 Requirements and Recommendations](#). The Device is expected to conform to the [OCP Hardware Secure Boot V1.0](#). This includes all images including recovery. A common glossary of terms for OCP security is referenced [here](#).

4.2 Device Recovery and SP800-193 Alignment

The Device shall follow the guidelines defined in the NIST SP800-193 specification, for its firmware protection, tamper detection and recovery capabilities.

A Device may or may not be able to persist critical data through a recovery process. A Device that is unable to maintain critical data MUST go through a process to reestablish this data or be returned to the manufacturer.

The Recovery process is a critical process for the overall security of the platform. This document focuses on Device recovery, specifically of the AC-RoT which is a symbiont Device to the PA-RoT. Device recovery's goal is to return a Device to its normal operation state running the correct firmware as verified by Device Attestation.

4.3 Detection, remediation, corruption and initiating recovery action

The primary mechanism of detecting the boot state of symbiont Devices by PA-RoT shall leverage reporting and attestation capabilities of Devices primarily based on the OCP Attestation specification.

If a Device does not pass attestation, then remediation must occur. If the Device is functional and has trusted images (healthy firmware and can respond to attestation), the firmware should be updated via standard means. An example of firmware update procedures are described in DMTF PLDM (Platform Level Data Model) Firmware update specification. If the Device is sufficiently out of date, a PA-RoT can choose to recover the Device and to return it to a consistent state.

Date: Dec 19, 2025

This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](#).

This document further proposes a protocol to create a standard recovery protocol across managed symbiont Devices. In case of critical failure resulting in the symbiont Device not being able to communicate over a high-level protocol, the Device and PA-RoT shall fallback to recovery via one of the supported recovery interfaces.

4.3.1 Types of Failures

Firmware recovery may be required for various reasons, the critical one being a corrupted firmware image leading to boot failure. The attestation capabilities defined in the OCP Attestation document provide the mechanisms to help identify issues with later stages of Device boot and configuration such as non-compliance of either firmware version or configuration, but do not directly help reporting critical boot failures. Two types of remediation are defined below:

- Update: Device is in a functional state, but has old firmware or configuration. This is updated via standard means and the security state is validated via attestation.
- Recovery: The Device may or may not be in a functional state. The Device can enter this state via internal error handling mechanisms or the PA-RoT can force recovery if permitted. The mechanism for Recovery is the primary goal of this document.

4.3.2 Response to firmware/configuration failures

Depending upon the failure one of the following actions can be performed:

- The Device fails attestation. Standard firmware update mechanisms can be used to bring the Device into compliance.
- The Device detects a failure and voluntarily enters recovery. This recovery interface can be used to download or select a recovery image.
- The Device is unresponsive (e.g. does not respond to MCTP messages). Forced recovery can be used in this case to recover the Device. If force recovery is not enabled, the Device MAY need to be returned to the manufacturer.

Upon detection of firmware code or critical data corruption, PA-RoT SHALL initiate the recovery process, which MAY be gated by authorization from the system administrator.

4.3.3 Administrative Forced Recovery

The RA instructs the Device to enter recovery mode. This can be achieved using a recovery command or physical presence indication (e.g. GPIO). The feature can be administratively disabled but SHOULD be enabled by default. The control enabling this is outside of the scope of this document. This can be used to bring a Device back into compliance with minimal reliance on state of the Device. This command forces the Device into a recovery state regardless of the initial Device state. Note: Forced recovery implicitly trusts the RA/PA-RoT. This can cause a denial-of-service attack affecting the Device availability by using this interface to repeatedly resetting/recover the Device.

Date: Dec 19, 2025

This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

4.4 Recovery requirements across platform components

A recovery process might require recovering several Devices or components to ensure a base security state. By performing recovery across all critical components on the platform to a known good set of firmware code, the PA-RoT ensures a recovery to a consistent and known state.

In order to provide long term security objectives, the PA-RoT should provide a means of updating recovery images, since relying on a golden image (static recovery installed at manufacturing) for recovery can lead to roll back to possibly vulnerable firmware. An example method of updating a recovery image would be through A/B versioning, where one version is latest/active, while the second acts as recovery. In this scheme an update is performed to the recovery version, upon completing the update the latest/active and recovery roles switch between A/B versions. In such A/B versioning schemes, this ensures that the recovery image is not older than the N-1 version. If both A and B copies are corrupted, then a dedicated recovery image (C-image) is used. If kept locally, this image should have the minimal footprint to enable attestation and firmware update to bring the Device into compliance. The C-image can also be pushed into the Device using the indirect memory interface described in this document. The pushed C-image is not required to support attestation since its role is transient. Note the C-image is signed with the same secure boot keys as the production images and is subject to the same anti-roll back rules.

Summary of images and roles

- A-image: A copy of the operational image.
- B-image: B copy of the operational image.
- C-image: Recovery image used to install/update A or B copy
- Critical data: data which is critical to the security of the Device (e.g. provisioned identity)

4.5 Recovery Authorization

The platform policies may require an authorization of the recovery process. This can be implemented through requiring physical presence indication, GPIO connected to the Device (AC-RoT/PA-RoT) or through forced recovery interface. Once the PA-RoT has received authorization, the recovery of the platform Devices (AC-RoT) can proceed. The PA-RoT can force a recovery through this interface, if enabled, or can use a physical presence/GPIO to enter recovery. The forced recovery interface MAY be disabled via a Device specific means, but SHOULD be enabled by default. The PA-RoT SHOULD be capable of performing recovery on demand, at the behest of an authorized entity.

4.6 Reasons for Device Recovery

A Device may or may not be able to persist critical data through a recovery process. Devices unable to maintain critical data **MUST** go through a process to reestablish this data.

The following sections define the requirements for recovery of a Device to an approved state (approved by platform owner).

Device recovery flows for a component (AC-RoT) **SHALL** be initiated by the PA-RoT (RA) on the following conditions:

1. Tampering of Device firmware and/or sensitive security parameters is detected by the platform root-of-trust.
2. Device is in an unknown state/fails to respond.
3. Forced recovery
4. Device detects corruption and enters recovery
5. PA-RoT determines via dynamic means (e.g. SPDMA Challenge) the Device is out of compliance.

Tampering of Device firmware and/or sensitive security parameters **SHALL** be detected by OCP Attestation.

A PA-RoT **MAY** use platform specific mechanisms to isolate the Device until it can be brought into compliance. An example would be holding the PCIe® reset of a Device until it is properly recovered.

4.7 Device Recovery Use Cases

Out of the several targeted use cases for recovery, this section outlines three specific use cases. From a NIST SP800-193 perspective, critical data includes provisioned identity as well as security critical parameters. Depending on the type of failure, recovery may not include this critical data.

4.7.1 Device Software Update

In this case, the Device has been secure-booted properly (including anti-roll back check) and has a functional attestation agent compliant with the OCP Attestation specification. Upon successful challenge and comparison of measurements against the platform manifest, the Device firmware or configuration can be updated using unspecified update techniques. Examples of these update techniques are DMTF PLDM firmware update or Cerberus firmware update. In this case, the recovery process is not required or used.

4.7.2 Device Recovery with Critical Data

The Device has entered or was forced into recovery by the RA. If supported, the C-image can be enabled from Device flash to recover the Device. The C-image **MUST** be a dedicated firmware

Date: Dec 19, 2025

This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

used to bring the Device back into compliance. If the recovery image is not present or valid, this recovery image can be pushed into the Device using the recovery flow as described in this document. This use case assumes security information (critical data) and identity provisioning (i.e. provisioned certificate stacks) are still available in the Device.

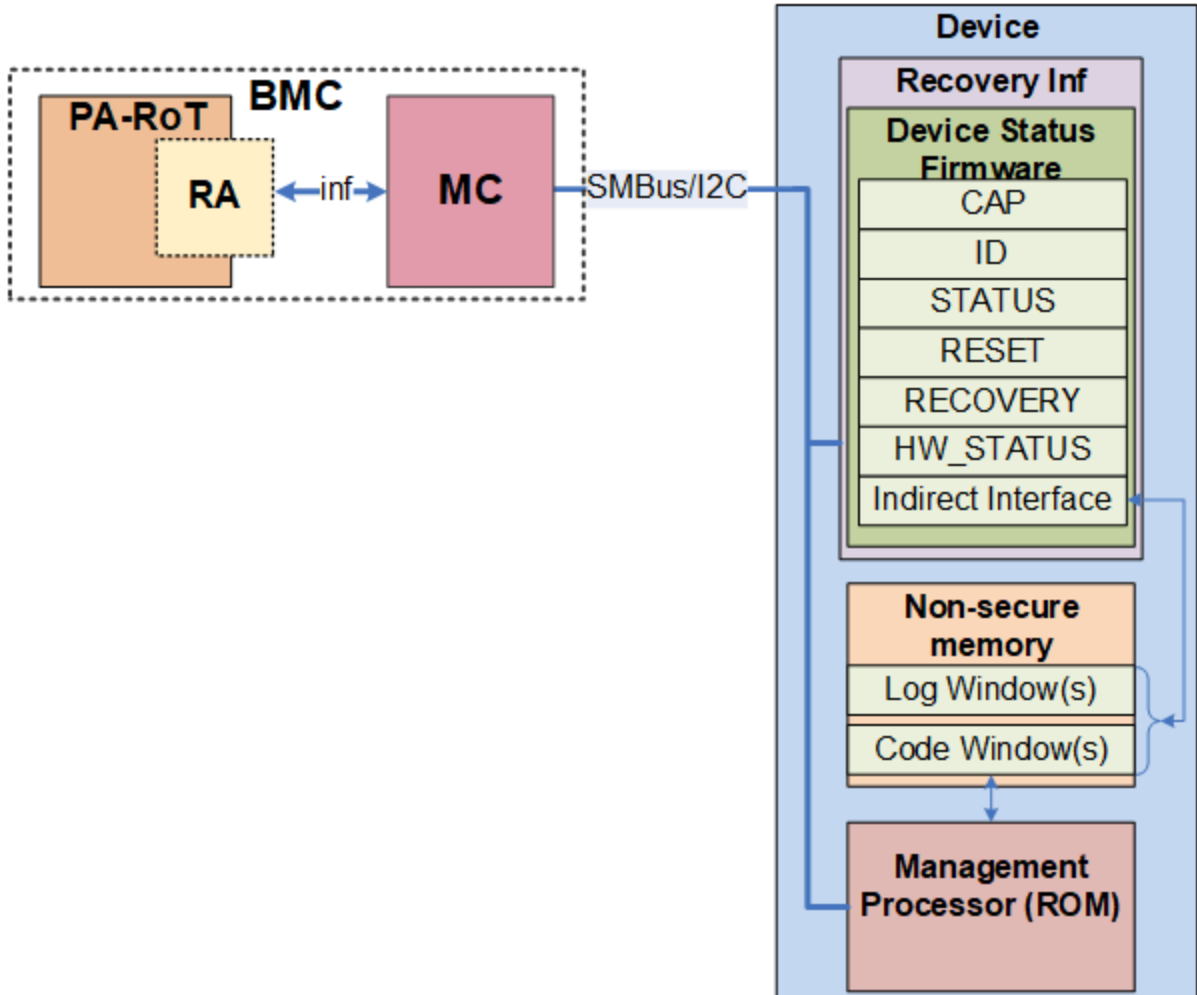
4.7.3 Device Recovery without Critical Data

The Device has entered or was forced into recovery by the RA and the critical data is corrupted or erased (NOTE: this cannot be used for ownership transfer). Critical data is defined in SP800-193 as “mutable data which persists across power cycles and must be in a valid state for the booting of the platform to securely and correctly proceed”. In this context, this can include key manifests, provisioned certificates used for attestation, etc.

The same recovery procedure described above with intact critical data applies. However, once the recovery image is booted the critical data needs to be reprovisioned into the Device. There are several means to reestablishing this identity including using a manufacturer provisioned certificate stack, if available. Depending on the characteristics of the Device, it may not be possible to restore the Device's critical security data without returning the Device to the manufacturer or provisioning facility.

5. Recovery Components and Roles

The following picture show various components of the recovery process



5.1 RA & PA-RoT Recovery Role

The RA & PA-RoT are responsible for the following tasks:

- Maintains the list of approved operational FW images for the Devices
- Maintains the list of approved recovery FW images for the Devices
- Protects the operational FW images from denial-of-service attacks
- Pushes operational FW images into the Device (include recovery images)
 - For Devices which store this recovery image locally the PA-RoT and AC-RoT MUST provide a mechanism for updating this firmware.
- Downloads recovery FW images into the Device
- Invokes Device recovery from either persistently stored recovery FW image or downloaded recovery FW image

5.2 AC-RoT Recovery Role

The AC-RoT SHALL perform the following tasks:

- Authenticates FW images during the Secure Boot process
- Authenticates ALL FW images via a cryptographic signature before usage
- Authenticates ALL FW images using key material that is cryptographically bound to immutable keys
- Authenticates downloaded recovery FW image that is transiently stored in RAM/DDR before usage
- Authenticates All FW images persistently stored inside the Device during Device SecureBoot process
- Enforces Rollback policies using FW version, Key Revocation, anti-rollback counter
- Reports Secure Boot failure error flags. MAY rely on the secure boot process to validate the image as long as it reports this failure via the Recovery Reason code “Missing/corrupt boot loader (first mutable code) firmware image” (BFFIMC)
- Supports Device Attestation identity authentication
- Includes ALL FW images (including persistently stored operational and recovery images) in the reported Device Attestation results
- Manage Device’s Recovery State

Immutable hardware SHALL serve as the Device root-of-trust for recovery purposes. This can be implemented in Devices as hard-coded logic and/or immutable ROM code. Depending on the corruption, the Device's cryptographic identity may not be available until recovery has been completed.

The Device SHALL provide a path for the RA to push a recovery image or select a recovery image from a local source.

Post-recovery, standard Out-of-Band (OOB) Attestation should be performed to verify the Device compliance. In case of recovery, it is possible that the security critical data or provisioned information (i.e. owner’s certificate) may have been lost, in such case attestation may fail. In such scenarios the Device may require re-provisioning of this information or return to the manufacturer.

As part of the recovery flow, the Device shall provide a mechanism to replace the current mutable operational firmware image with an approved version that shall be maintained in the PA-RoT. The Device SHALL determine validity of the approved version by performing digital signature verification of the firmware according to the OCP Secure Boot specification. This implies that the recovery image is signed by the same RoT as the operational firmware and uses the same anti-roll back counters.

Date: Dec 19, 2025

This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

5.3 Recovery Image

The PA-RoT is responsible for maintaining the list of approved firmware for Devices and protecting the recovery images from denial-of-service attacks. The RA is responsible for pushing or activating this image onto the Device. In the case of a recovery image, the PA-RoT SHOULD maintain a list of recovery images for the Device. For Devices which store this recovery image locally it MUST provide a mechanism for updating this firmware.

The recovery image should be considered an operational image and MUST follow the OCP secure boot document. Specifically, this image will be signed with the firmware signing key and is subject to the anti-rollback counters. The Device SHOULD verify the cryptographic signature of the recovery image as a pre-step before performing recovery. A compliant Device MAY rely on the secure boot process to validate the image as long as it reports this failure via the Recovery Reason code.

The following table is a summary of the firmware components.

TERMINOLOGY	DEFINITION
A-Image	A copy of operational image with associated Critical Data[x]. This image is persistently stored in Device's flash memory.
B_Image	B copy of operational image with associated Critical Data[y]. This image is persistently stored in Device's flash memory.
C_image:PersistentRecoveryImage	This FW image installs/updates the A/B images This image is persistently stored in the Device's flash memory. This image is selected via recovery image selection
C_image:TransientRecoveryImage	This FW object installs/updates the A/B images This image is transiently stored in the Device's transient memory. This image is selected via recovery image selection
FW_Update:NormalImage	This FW download package updates A/B and MAY contain C_image:PersistentRecoveryImage.

FW_Update:TransientRecoveryImage	This FW image is transformed into C_image:TransientRecoveryImage. This image is transiently stored in RAM.
All FW images shall support the following qualities: • shall support cryptographic authentication (FW signatures) • shall be signed with key material that is cryptographically bound to the Device's immutable ROOT Key (secure boot) • shall check for Rollback protection	

6. Recovery Process

When needed, the PA-RoT uses the recovery agent (RA) to perform the Device recovery flow. A recovery agent (RA) is a defined component which is responsible for performing the recovery process. The RA MAY be part of the PA-RoT or a separate component. The RA will use the PA-RoT as the source for all images and configuration. A multi-state recovery process may be required in order to bring the Device into full compliance.

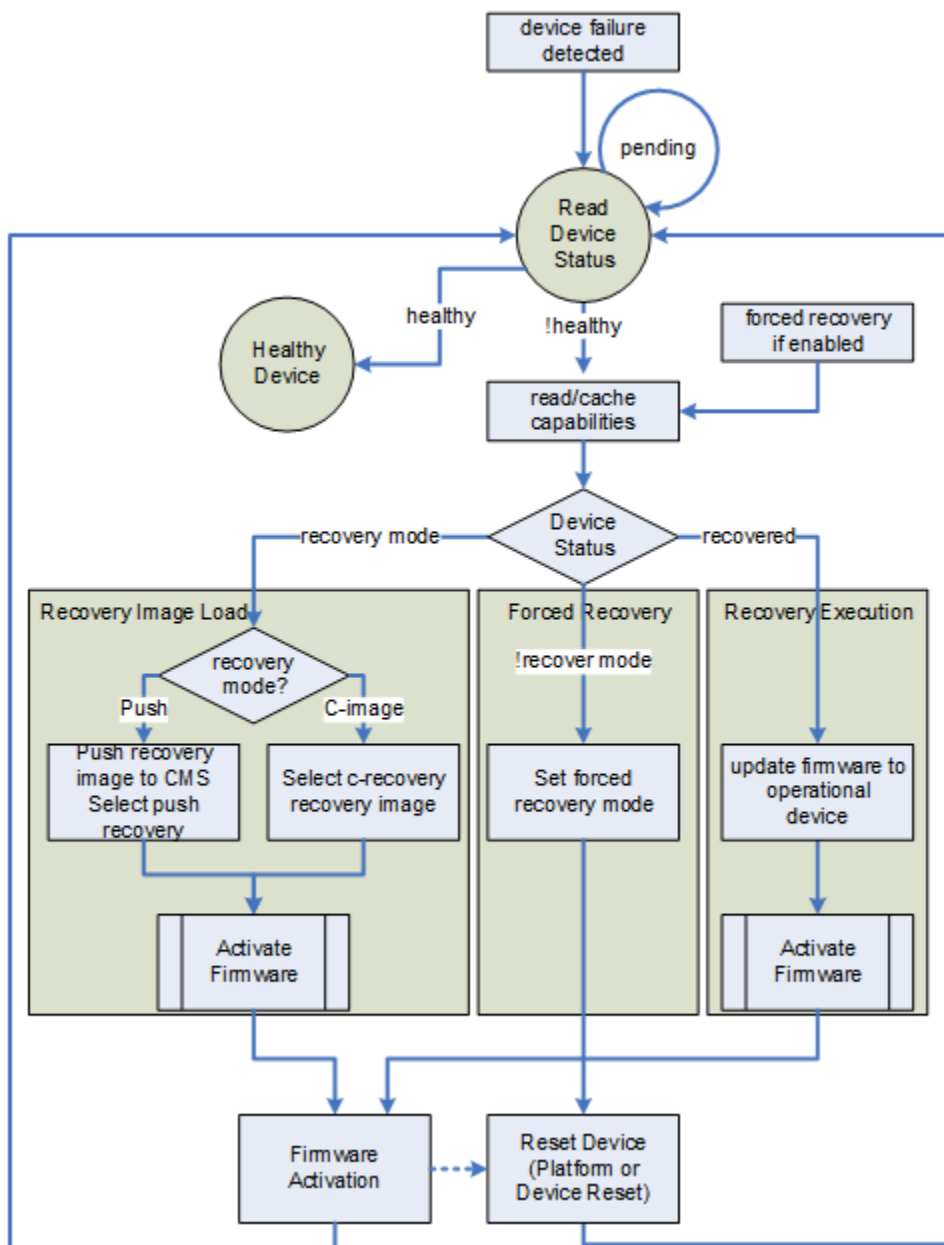
The health of a Device is maintained by the platform and is outside of the scope of this specification. This determination is typically done by using information from the PA-RoT. If a Device is declared unhealthy, normal software update procedures (e.g., DMTF PLDM) should be used to bring the Device into compliance. That is, the recovery process is not a replacement for a normal software update and should be viewed as a last resort before the Device is declared un-recoverable.

The RA can query the Device status using this protocol via reading the Device status register. During operational boot, the Device status MUST indicate status for the software component which is used for OCP attestation. Other software components' status (e.g. other compute domains within a Device) SHOULD NOT be reported in this status register. The status register is only valid when it is not zero (e.g. not pending). This allows time for a boot process on the Device to properly reflect the status. This status is informational only and the state of the Device SHOULD be cryptographically attested by using the attestation procedure in the OCP attestation documents.

The following picture depicts the overall process to recover a Device. The recovery process is entered after a Device is declared unhealthy by the platform or the Device. A Device MAY be administratively commanded to recover via forced recovery, if enabled. Forced recovery MAY be disabled via a Device specific means. A Device MUST advertise forced recovery via the capabilities described in this spec if a Device is capable and enabled. The Device status is read to determine the next steps of the recovery process.

Date: Dec 19, 2025

This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).



A Device passes through a number of states in the recovery process. The following definition of the defined states:

- **Healthy** - the Device is running an operational image. This state is designed only to reflect the status of the management entity firmware. Specifically, in Devices which contain the attestation agent and firmware update process. The PA-RoT is responsible for determining if the Device is healthy.
- **Not healthy** - the Device is not healthy. The recovery reason code contains additional information. Depending upon the recovery reason code, additional information can be optionally communicated.

Date: Dec 19, 2025

This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

- Recovery state - a Device in the recovery state is ready to accept either a pushed image or a command to use the recovery image stored on the Device (C-image).
- Recovery pending - a Device which has completed the image push or selected the recovery image.
- Recovery successful - The recovery image is currently running.

7. Recovery Interface Functions

7.1 Device Reset

Multiple Device resets may be required to fully recover a Device. A Device MAY support Device reset via this interface (RESET) or rely on a platform reset mechanism. A Device MAY support two different types of resets. A Device reset via the RESET registers will reset the Device and MAY cause a Host bus enumeration. It MAY cause an SMBus reset. A Device MAY support a management reset where only a subset or management portion of the Device is reset. A Device which supports management reset:

- MUST NOT cause a bus re-enumeration of the Device.
- MUST reset all security components of the Device. This includes any processor subsystem responsible for attestation of the Device.

7.2 Forced Recovery

A Device can be commanded to enter recovery mode. This is achieved by writing forced recovery to the RESET register. At the next reset, the Device will enter recovery mode. A Device MAY disable forced recovery via Device specific means. A Device which has been commanded to enter recovery but forced recovery is disabled MUST report “Error entering Recovery mode” in the RECOVERY_STATUS register.

7.3 Flashless Boot

A Device can be commanded to enter flashless boot mode. This is achieved by writing flashless boot to the RESET register. At the next reset, the Device will enter flashless boot mode. The purpose of flashless boot mode is to enable the PA-RoT to boot more than just the recovery image via the recovery interface. This usage model is explained in the OCP whitepaper [\[EXTERNAL\] Flashless Boot using OCP, PCIe®, and DMTF Standards](#). Support for flashless boot is an optional capability a Device MAY implement.

7.4 Recovery Image Push

A Device MAY have entered the recovery state based on local error conditions. For example, a signature failure on the first mutable image loaded by the ROM. The Device MUST reflect the recovery state in the Device status register. A Device in this state is ready to receive or select the recovery image.

A Device in recovery is ready to accept the recovery image if supported. This image is pushed via the memory window or a local recovery image can be selected. This selection is written to the RECOVERY_CTRL register. If the push protocol is used it MUST be written using the indirect memory protocol to a memory region (CMS) specified for code. The recovery window CMS MUST be selected in the RECOVERY_CTRL command. The RECOVERY_CTRL command MUST be completed before image activation.

7.5 Recovery Image Selection

A Device which supports c-image as the recovery image MUST use the RECOVERY_CTRL to select this mode. This command MUST be completed before image activation.

7.6 Recovery Image Activation

Once the image is fully written or local c-image is selected, an activation command MUST be performed to activate the recovery image. The activation process MUST ensure the Device restarts into the Device immutable trust anchor as described in OCP secure boot document. An image waiting for activation of the recovery image MUST report “Recovery Pending” in the DEVICE_STATUS. The result of activation will start running the recovery image. A Device MAY use a management reset to implement the activation function.

A Device may support multiple recovery image stages. To facilitate loading the next image, the Device MUST report “Awaiting recovery image” in the RECOVERY_STATUS register. The RA can then load or select the next recovery image and trigger activation. The result of activation will start running the next recovery image stage. The Device shall set the RECOVERY_STATUS to “Bootting recovery image”. The Device SHALL set “Recovery failed” in the RECOVERY_STATUS register if any recovery image stage fails to activate. If the image activation is successful, and another recovery image stage is expected, the Device shall increment the “Recovery image index” and set the RECOVERY_STATUS to “Awaiting recovery image”. If no other stages are expected, the Device shall set the RECOVERY_STATUS to “Recovery successful”.

NOTE:

It is possible to combine image selection and image activation. In this case, the recovery image and activation are sent in the same command RECOVERY_CTRL (i.e. writing

Date: Dec 19, 2025

This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

0xF|imagemode|CMS). This will cause the Device to immediately start executing the recovery image.

7.7 Recovery Image Authentication and Operation

After the image is activated, it executes as an operational image and **MUST** pass all security checks defined in the OCP secure boot document. The recovery image is bringing the firmware or configuration data into compliance. A local C-image **MUST** additionally support OCP attestation. Once the recovery process is complete, the Device **SHOULD** use Device reset to activate an operational image.

7.8 Normal/Healthy Operation

A Device in this state is fully functional and running operational firmware. The Device **MUST** report “Device Healthy” in the DEVICE_STATUS registers when running operational firmware.

8. Recovery Interface

The recovery interface abstractly is described by block read and write commands. These commands can be implemented using many protocols. The following sections describe different mechanisms.

The following command groups are defined in the following table:

Table 1 - Recovery Command Summary			
Command (decimal, hex)	Req	Scope	Notes
PROT_CAP (34, 0x22)	Y	A	Device Capabilities Information
DEVICE_ID (35, 0x23)	Y	A	Device identity information
DEVICE_STATUS (36, 0x24)	Y	A	Device status information
DEVICE_RESET (37, 0x25)	N	A	Device reset and control
RECOVERY_CTRL (38, 0x26)	Y	A	Recovery control and image activation
RECOVERY_STATUS (39, 0x27)	N	A	Recovery status information
HW_STATUS (40, 0x28)	N	R	Hardware status including temperature
INDIRECT_CTRL (41, 0x29)	N	R	Indirect memory window control
INDIRECT_STATUS (42, 0x2A)	N	R	Indirect memory window status

INDIRECT_DATA (43, 0x2B)	N	R	Indirect memory window for pushing recovery image
VENDOR (44, 0x2C)	N	R	Vendor-defined behavior
INDIRECT_FIFO_CTRL (45, 0x2D)	N	R	Indirect FIFO control
INDIRECT_FIFO_STATUS (46, 0x2E)	N	R	Indirect FIFO status
INDIRECT_FIFO_DATA (47, 0x2F)	N	R	Indirect FIFO write aperture

The req column indicates if the command is required. The scope column indicates when the command must be active (e.g., RA can expect a response).

- A - indicates the command should be available anytime the Device recovery interface is available.
- R - indicates the recovery interface must be active. This is indicated by a non-zero DEVICE_STATUS.

8.1 Capability/Discovery

The capabilities of the Device are discovered via reading PROT_CAP.

8.2 Indirect Memory Interface

A common indirect access mechanism is defined to facilitate reading and writing memory spaces within the Device. Component Memory Spaces (CMS) are mapped directly to the resources within the Device in a Device specific way or via a FIFO approach. The Device resource can be memory, registers, flash or other Device resource. This interface allows for a common interface to exchange code, logs or other vendor defined data with the Device. If the indirect memory interface is supported, it MUST support at least one memory region. The code, critical and vendor-defined CMS types are defined and described in the following sections.

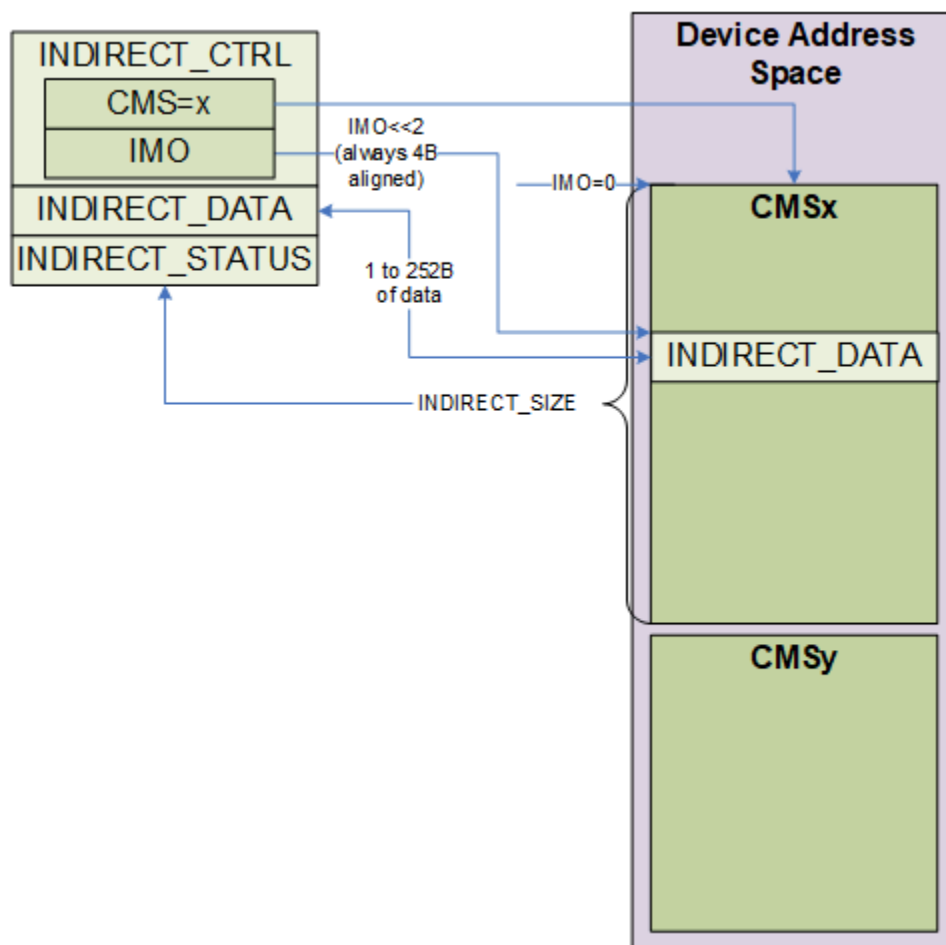
The size (INDIRECT_SIZE) and type (INDIRECT_TYPE) of memory space is queried by writing the memory region to the INDIRECT_CTRL register and reading the INDIRECT_STATUS. A CMS can use either polling or direct access. This is reported as the high-order bit of the CMS type.

8.2.1 Addressing within a Component Memory Spaces

Addressing within a CMS maps the INDIRECT_DATA window into the CMS. An indirect memory offset (IMO) is maintained within the current CMS and is always 4-byte aligned. The base address of the CMS within the Device is vendor defined. Before writing to a CMS, the type and size of a

CMS SHOULD be determined by the INDIRECT_CTRL command. The first byte of an address region is defined as byte 0 written to the INDIRECT_DATA registers when the indirect memory offset (IMO) is zero. The IMO is incremented by the number of bytes written to the INDIRECT_DATA registers rounded up to the next 4-byte boundary. Changing the CMS in the INDIRECT_CTRL MUST reset the IMO to a value within the CMS region. For non-polling regions, the Device is expected to be able to accept continuous requests. If a region requires polling, the ACK status is reported in the INDIRECT_STATUS register when the Device can accept the next transaction. The RA MUST poll the INDIRECT_STATUS register before the next INDIRECT_DATA transaction. The ACK indication is cleared on read.

The following figure shows the relationship between various indirect commands, the CMS and Device resources.



Error conditions

- Address Wrapping: (e.g. the IMO extends beyond the reported size). This MUST wrap to the beginning of the buffer AND report an overflow in the **INDIRECT_STATUS**.
- Writing to a read only CMS: This MUST not write to the internal address space and report an access error in the **INDIRECT_STATUS**.

Date: Dec 19, 2025

This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

- Writing or Reading to a polling enabled region when not ready. This transaction MUST be ignored and MUST NOT increment the IMO. A polling error in the INDIRECT_STATUS MUST be reported.
- Unaligned access to CMS. IMO address will be truncated (e.g. lower bit set to zero) and accepted without generating error.

8.2.2 Code CMS

A code region is designed to deposit code (the recovery image) to facilitate the recovery process. A Device which supports code push MUST support at least one memory region and be mapped to Memory region 0 (CMS=0). Multiple code spaces can be used to support multiple domains within the Device, but these are used in a vendor specific way.

The image can be activated by writing the recovery CMS to the RECOVERY_CTRL register. For these spaces, care MUST be taken to prevent time-of-check to time-of-use attacks. One way to accomplish this is to map the code region to a non-secure region and close the region after it has been activated.

8.2.3 Critical Logging CMS

This CMS used for logging is defined as read only. Support of critical logging is optional. Write access to the region MUST not make changes to the CMS and report an error in the INDIRECT_STATUS register. The critical logs are not signed and no security guarantees are provided. In addition, the logs may not persist a device reset.

The log is viewed as a circular buffer. Entries are added to the log in a sequential fashion. The entry identifier must be unique and monotonic. Based on parsing of the debug log and entry identifiers, the event sequence can be inferred. The general structure uses a magic number, length and entry_id. The entry contains a debug log entry which contains a format and opaque payload.

```
LOG_MAGIC_NUMBER 0xE5E5
struct debug_log_entry {
    struct logging_entry_header header; /**< Standard logging header. */
    struct debug_log_entry_info entry; /**< Information for the log entry. */
};
struct logging_entry_header {
    uint16_t log_magic;                /**< Start of entry marker. */
    uint16_t length;                   /**< Total Length of the entry. */
    uint32_t entry_id;                 /**< Unique entry identifier. */
};
struct debug_log_entry_info {
    uint16_t format;                   /**< Format of the log entry (msg_body) */
    uint8_t msg_body[];                /**< body of log message. */
};
```

Date: Dec 19, 2025

This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

8.2.4 Vendor Defined CMS

Two types of vendor defined regions are defined, one which is read only (vendor defined logs) and one which is read/write.

8.2.5 Indirect FIFO CMS

INDIRECT_FIFO_CTRL, INDIRECT_FIFO_STATUS, and INDIRECT_FIFO_DATA registers are an interface to an alternative FIFO model. The FIFO model enables higher speed transfer because the Recovery Agent can repeatedly access INDIRECT_FIFO_DATA without checking the status. The disadvantages of the FIFO model is that it does not allow random access and only allows one type of transaction per instance: reads or writes.

PROT_CAP command indicates support for FIFO CMS. A given CMS index shall be either a memory window CMS or a FIFO CMS. INDIRECT_STATUS and INDIRECT_FIFO_STATUS commands can be used to determine the CMS type given the CMS index programmed into INDIRECT_CTRL / INDIRECT_FIFO_CTRL respectively. The RA shall only access INDIRECT_CTRL/INDIRECT_STATUS/INDIRECT_DATA if PROT_CAP BIT 5 is set. The RA shall only access INDIRECT_FIFO_CTRL/INDIRECT_FIFO_STATUS/INDIRECT_FIFO_DATA if PROT_CAP BIT 12 is set. If PROT_CAP BIT 5 is set but the CMS index programmed into INDIRECT_CTRL does not support the access type, INDIRECT_STATUS shall return “Unsupported Region”. If PROT_CAP BIT 12 is set but the CMS index programmed into INDIRECT_FIFO_CTRL does not support the access type, INDIRECT_FIFO_STATUS shall return “Unsupported Region”.

To use the FIFO CMS, the RA:

1. Programs the CMS index, reset, and image size fields in INDIRECT_FIFO_CTRL.
2. Reads INDIRECT_FIFO_STATUS to see the FIFO size, max transfer size, and type of region.
3. Issues transfers to INDIRECT_FIFO_DATA with size less than or equal to the max transfer size and the image size remaining.

The Device shall return a NACK response for any transfer that would cause the Write Index to advance to equal the Read Index. The RA can implement flow control through any of the following options:

1. NACK responses
2. Monitoring the FIFO space remaining via the Write Index and Read Index.
3. Polling on the FIFO empty status bit.

FIFO space remaining is computed as

$$(\text{Read Index} - (\text{Write Index} + 1) + \text{FIFO Size}) \% \text{FIFO Size}.$$

FIFO occupancy is computed as:

Date: Dec 19, 2025

This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

$(\text{Write Index} - \text{Read Index} + \text{FIFO Size}) \% \text{FIFO Size}$.

8.3 Recovery SMBus Interface

This section describes a recovery protocol based on SMBus/I2C block read and write commands. The recovery protocol requires an SMBus compliant interface between RA and the managed Device used to transport the recovery protocol. The recovery protocol is designed to be simple and MUST be embedded into ROM or dedicated hardware.

As per SMBus 3.1 spec, each Block Write/Read contains a byte for the Command Code, followed by a byte for Byte Count and up to 255 bytes of data. Command data sizes vary depending on the commands; refer to table below for specifics.

The following are the requirements for the recovery interface for SMBus/I2C:

- MUST be compliant with "System Management Bus (SMBus) Specification version 3.1, 19 Mar 2018".
- MUST support physical layer per spec specified in [SMB 31]
- MUST support Class 100 kHz operation
- SHOULD support Class 400 kHz and Class 1 MHz operation
- MUST support data link layer per spec in [SMB 31]
- MUST support network layer per spec in [SMB 31]
- MUST support block read and write
- SHOULD support and use Address Resolution Protocol (ARP) for dynamic target address assignment.
- SMBus interface and recovery agent SHOULD be designed to have maximum uptime and have minimal external dependencies (e.g., flash).
- MUST respond to recovery commands sent by RA once target address is assigned or to default address
- MUST support target functionality
- MUST not support master functionality when in recovery mode
- MUST support a fixed I2C address, if ARP is not used. If a shared topology is used then the default SHOULD be 0xD4 (7-bit address including the lsb read/write bit) for a separate topology it SHOULD be 0xD2 (7-bit address including the lsb read/write bit).
- SHOULD use PEC checksum per SMB 3.1

The recovery protocol does not require MCTP or any variant of protocol that runs on top of MCTP. The recovery protocol does not depend on bidirectional communication initiation. The recovery protocol MAY exist on the same interface as the one used for MCTP, provided there is a way to deliver non-MCTP commands to the recovery interface. The recovery interface can also be a separate standalone SMBus interface. The following two diagrams depict a few different

topologies. NOTE: The preferred topology is a separate address for the recovery interface. I3C will only support this topology since the command byte is not specified in the I3C specification.

8.3.1 SMBus Topology

There are two topologies supported. In the first topology, the recovery interface and the MCTP EP share the same SMBus address. Note in this topology the default I2C address SHOULD be 0xD4.

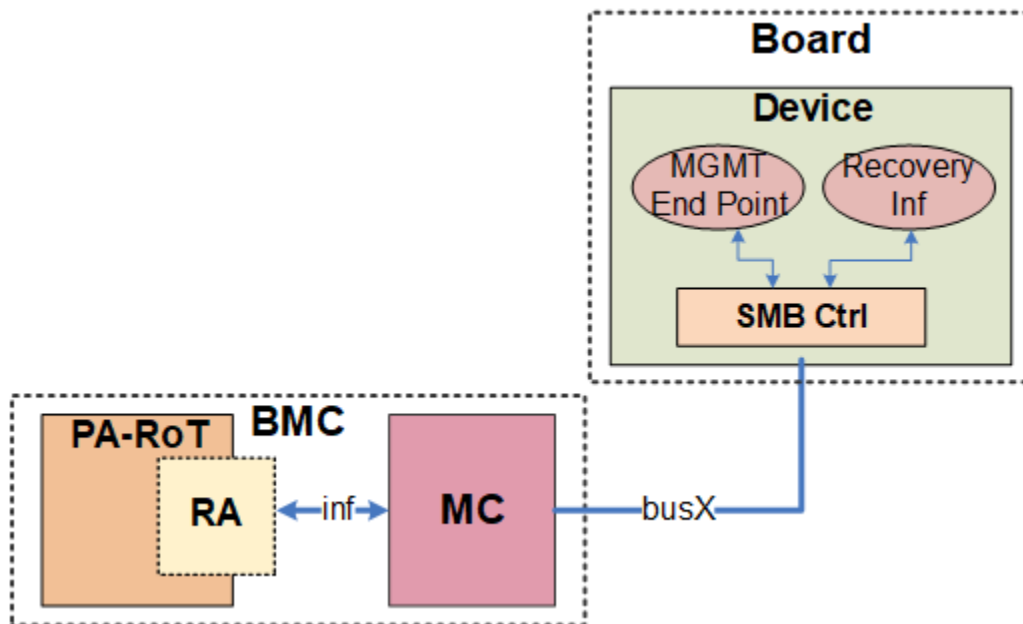


Diagram of shared SMBus interface

The second topology has separate components for the recovery interface and MCTP End Point (EP). Note the connection between the controllers can be external (e.g. separate pins) or internal. Note2 in the topology the default I2C address SHOULD be 0xD2.

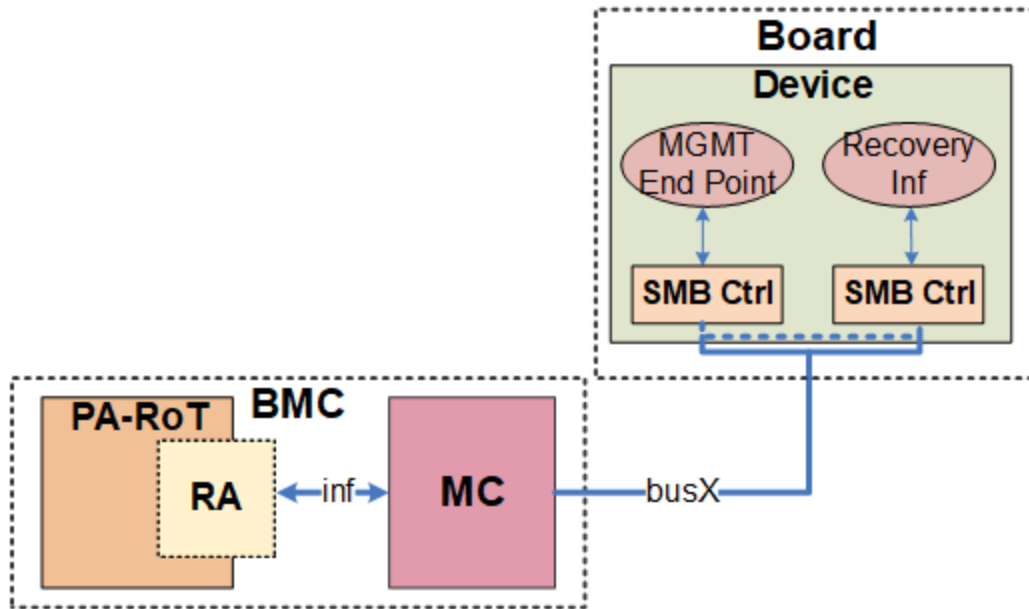


Diagram of separate SMBus interface for recovery

8.3.2 SMBus Device Addressing and Commands

The SMBus protocol uses a 7-bit Device addressing and will be used for recovery. In addition, an 8-bit command byte is defined in the block read/write commands. A compliant Device **MUST** support the required commands using SMB block read and write interface.

Several other protocols/standards which use block read and write commands for various functions. The following is a list of considered standards.

- DMTF MCTP over SMBus transport (v 1.0.0)
 - This specification reserves command 16 (0x0F)
- NVMe-MI v1.2 Out-of-Band (OOB) management
 - This specification reserves commands 0, 8 and 32
- OCP NVMe Cloud SSD Specification (v1.0)
 - Section 10.2 command 0, 8, 32, 50, 90, 96, 154, 242, 248

This specification does not overlap or conflict with command allocation from aforementioned standards. Therefore, a compliant Device can implement the recovery interface and be compatible with aforementioned standards on the same interface. Care must be taken to ensure future compatibility.

8.3.3 Interface Sharing/Isolation

SMBus is a multi-master protocol without fair arbitration. Device firmware could cause a denial-of-service to the recovery interface by mastering transactions that win arbitration in perpetuity. This would prevent the RA from issuing a Device reset or forced recovery commands.

Date: Dec 19, 2025

This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

To mitigate this a Device SHOULD disable SMBus mastering out of power-on. Bus Mastering can be enabled via Device specific means OR by using the interface master enable.

Devices that support interface isolation MUST report the capability in the PROT_CAP command and ensure mastering is disabled as part of the Device power-on reset sequence. The RA MUST enable mastering, in the DEVICE_RESET register, when it determines the Device is healthy. Note: MCTP notification will fail if mastering is not enabled.

8.4 Recovery I3C Interface

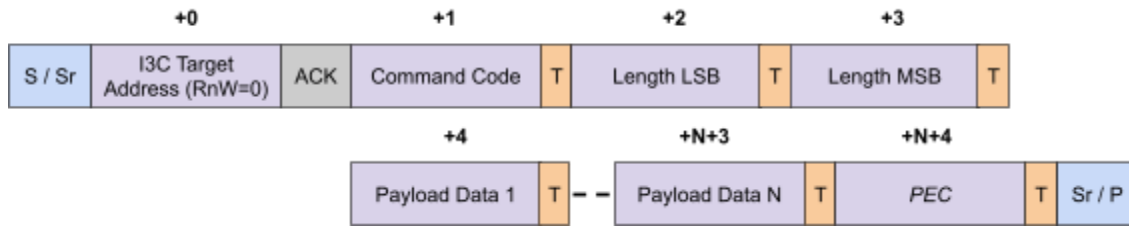
The recovery interface defines a protocol-agnostic command set that can be implemented over SMBus, I3C, or USB transport layers. Platform implementations MAY support any combination of these transports while maintaining command-level compatibility and consistent recovery functionality across all supported interfaces. This chapter defines the recovery interface over the I3C protocol. When multiple recovery interfaces are implemented, the Recovery Agent (RA) shall select and use only one bus protocol (SMBus, I3C, or USB) per recovery session, as determined during power-on reset discovery.

The following are the requirements for the recovery interface for I3C:

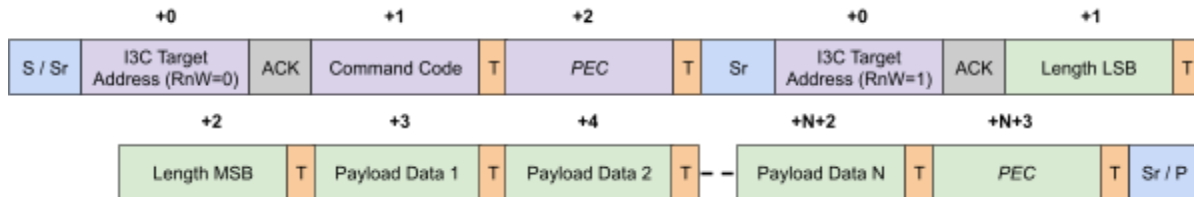
- MUST be compliant with "MIPI I3C Basic version 1.1.1".
- MUST support SDR mode of operation.
- MUST support target functionality.
- MUST support I3C Read Transfer and I3C Write Transfer.
- MUST support max read data length and max write data length advertised via GETMRL and GETMWL. Minimum MRL is 67 bytes and minimum MWL is 68 bytes.
- MUST not support initiator functionality when in recovery mode.
- MUST use dynamic addressing via ENTDA.
- MUST have a unique address compared to any DSP0233 MCTP I3C endpoint.
- I3C interface and recovery agent SHOULD be designed to have maximum uptime and have minimal external dependencies (e.g., flash).
- MUST respond to recovery commands sent by RA once target address is assigned
- MUST use PEC checksum as defined in DSP0233 MCTP I3C.
- MUST use DCR value of 0xBD

The command encoding is illustrated below. I3C does not support Block Writes and Block Reads natively, so this specification constructs equivalent behavior using Private Write Transfers and Private Read Transfers. The first bytes transferred are the address, command code, and length of the data for the subsequent transfer.

The bytes included in the PEC calculation mirror the definition in DSP0233 section 5.3.1. The PEC terminates a transfer prior to a Stop or Repeated Start condition. The PEC calculation excludes Start, Repeated Start, Stop, 7'h7E broadcast address, T-bits, ACK, and NACK. For write transfers, the Device shall verify the PEC byte correctness. If it detects an error, it should discard the write transfer and set the CRC error bit in DEVICE_STATUS. For read transfers, the RA shall verify the PEC byte correctness. If it detects an error, it should discard the read transfer.



Recovery I3C Interface Block Write



Recovery I3C Interface Block Read via Private Write and Private Read
Purple is Initiator-driven, Green is Target-driven

8.4.1 I3C Interface implementation notes

For configurations implementing both I3C and legacy I2C Devices on the same shared bus the following constraints apply:

- In a Mixed bus configuration, the maximum data rate as specified in I3C specification is possible only if all I2C Targets have the 50ns spike filter.
- In the absence of spike filters or if the presence of a filter is unknown, the maximum data rate is limited to only Fast Mode (400kHz) or Fast Mode Plus (1MHz), even for I3C Devices as per the I3C Specification. I2C Targets are not allowed to extend the clock.

RECOMMENDED is to use a pure I3C bus for high-speed critical recovery paths whenever possible. If a mixed bus configuration is unavoidable due to backward compatibility or legacy sensor requirements, perform mandatory pre-deployment validation including address conflict checks, verification that I2C clock stretching is disabled, confirmation that 50ns spike filters are enabled on all I2C Devices, and consideration of isolating or disabling I2C Devices that cannot meet these requirements.

8.5 Recovery USB Interface

The recovery interface defines a protocol-agnostic command set that can be implemented over SMBus, I3C, or USB transport layers. Platform implementations MAY support any combination of

Date: Dec 19, 2025

This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

these transports while maintaining command-level compatibility and consistent recovery functionality across all supported interfaces. This chapter defines the recovery interface over the USB protocol over Control Endpoint 0 (EP0). When multiple recovery interfaces are implemented, the Recovery Agent (RA) shall select and use only one bus protocol (SMBus, I3C, or USB) per recovery session, as determined during power-on reset discovery.

The following are the requirements for the recovery interface for USB EP0:

- MUST implement USB specification compliance for version 1.1 or later with guaranteed forward compatibility across all current and future USB host controller generations.
- MUST support standard USB Device operation mode (Device functionality) across all supported USB versions.
- RECOMMENDED USB Device speed is USB 2.0 High Speed or higher for optimal OCP Recovery flow operation.
- MUST support USB Device functionality with Control Endpoint 0 (EP0) as defined in USB 1.1, or later specifications.
- MUST support USB Control IN Transfer and USB Control OUT Transfer via control transfer transactions with version-appropriate timing and electrical characteristics.
- MUST support maximum control transfer data payload length via wLength field in SETUP packet, zero or more DATA packets and STATUS packet.
- OCP Secure Firmware Recovery MUST define wMaxWrTransferSize and wMaxRdTransferSize in the functional descriptor to provide supported Read and Write transaction sizes for data with the following requirements:
 - Maximum supported data payload size MUST be at least 64 bytes for both reads and writes ($wMaxRdTransferSize \geq 64$, $wMaxWrTransferSize \geq 64$) to satisfy OCP Secure Firmware Recovery command requirements.
 - Maximum supported data payload SHALL NOT exceed USB EP0 control transport limitations for the specific USB version, whose maximum is currently 65535 bytes limited by the wLength value.
- OCP Secure Firmware Recovery command payload transfer MAY require multiple DATA0/DATA1 packets for larger commands:
 - USB 1.1: 8, 16, 32 or 64 bytes per packet
 - USB 2.0 High Speed: 64 bytes per packet
 - USB 3.0 SuperSpeed and later: 512 bytes per packet
- All DATA packets SHALL be maximum packet size except the final packet. The final packet may be shorter than wMaxPacketSize or zero-length (ZLP) to indicate DATA stage completion per USB specification requirements.
- MAY NOT support other USB interfaces when in recovery mode even if the USB descriptor lists all supported interfaces. This specification allows for static USB descriptors with all interfaces defined. Other interfaces MAY respond with STALL to

all interface-specific requests until the initial recovery code is initialized allowing partial or complete firmware load or recovery using OCP Recovery flow while other interfaces are not fully operational. Typically, that limitation applies to ROM code and/or boot loader.

- MUST use standard USB Device enumeration process including USB Device descriptor, configuration descriptor, and interface descriptor exchange compatible with USB 1.1 and later host controllers.
- USB Device MUST be implemented as an USB Composite Device to implement OCP Recovery Interface over USB. bInterfaceClass = 0xEF (Miscellaneous) are allowed only on Interface descriptors and not in the device descriptor.
- MUST use bInterfaceClass = 0xEF (Miscellaneous) with a unique bInterfaceSubClass = 0x08 and bInterfaceProtocol = 0x01. Values are assigned by USB-IF to uniquely identify OCP Secure Firmware Recovery interfaces across all USB versions and distinguish them from other USB Device Interfaces see <https://www.usb.org/defined-class-codes>.
- USB interface and recovery agent SHOULD be designed to have maximum uptime and have minimal external dependencies (e.g., flash).
- MUST respond to recovery commands sent by RA once USB enumeration is complete and USB Device has entered the configured state, regardless of USB version. The OCP Secure Firmware Recovery interface over EP0 SHALL remain available during both recovery mode and normal runtime of USB Device operation.
- MUST implement USB control transfer integrity checking through standard USB CRC mechanisms and packet validation as defined in the respective USB specification versions (1.1, or later).

8.5.1 USB EP0 Command Encapsulation

Use the following format for OCP Secure Firmware Recovery over USB EP0 operations across all supported USB versions where single OCP Recovery command is mapped to exactly one EP0 control transfer:

SETUP - 1 Data OUT packet
wLength = actual number of bytes up to wMaxWrTransferSize as host controls write length

SETUP Token	Data0 (8 Bytes)	ACK
----------------	-----------------	-----

DATA - 0 or more DATA0/1 OUT packets
Device must accept optional ZLP if generated by host at the end of DATA transfer.

OUT Token	Data0/1 (Variable Bytes)	ACK/NACK/ STALL
--------------	--------------------------	--------------------

STATUS - DATA0 IN packet (ZLP)
Transferred bytes must match exactly wLength, otherwise STALL will be returned in STATUS by device.

IN Token	Data0 (0 Bytes)	ACK/NACK/ STALL
-------------	-----------------	--------------------

OCP Recovery over USB EP0 Write

SETUP - 1 Data OUT packet
wLength is always equal to wMaxRdTransferSize as device controls read length

SETUP Token	Data0 (8 Bytes)	ACK
----------------	-----------------	-----

DATA - 0 or more DATA0/1 IN packets
End of transfer is with last packet shorter than wMaxPacketSize or ZLP

IN Token	Data0/1 (Variable Bytes)	ACK/NACK/ STALL
-------------	--------------------------	--------------------

STATUS - 1 DATA0 OUT packet (ZLP)

OUT Token	Data0 (0 Bytes)	ACK/NACK/ STALL
--------------	-----------------	--------------------

OCP Recovery over USB EP0 Read

SETUP phase definition:

Offset	Field	Size	Value	Description
0	bmRequestType	1	Bit Mask	Bit 7 - Direction 0b - Write (USB Host to USB Device) 1b - Read (USB Device to USB Host) Bit 6:5 - Type 01b (USB Class)

Date: Dec 19, 2025

 This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

				Bit 4:0 - Recipient 00001b (USB Interface)
1	bRequest	1	0x00	0x00 - OCP RECOVERY TRANSFER 0x01..0xFF - Reserved
2	wValue[0]	1	Command ID	OCPRec Command ID
3	wValue[1]	1	0x00	Reserved
4	wIndex[0]	1	Interface ID	USB Interface ID - as defined in the USB descriptor
5	wIndex[1]	1	0x00	Reserved
6	wLength	2	Length	Length of data phase in the range 0..65535 bytes. For Write (USB Host to USB Device) wLength must be equal or lower than wMaxWrTransferSize as defined in the Interface's Functional Descriptor. For Read (USB Device to USB Host) wLength must be always set to wMaxRdTransferSize as defined in the Interface's Functional Descriptor to allow USB Device to control the Read length.

8.5.2 USB Interface Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	0x09	Size of this descriptor in bytes.
1	bDescriptorType	1	0x04	INTERFACE descriptor type.
2	bInterfaceNumber	1	Number	Number of this interface. RECOMMENDED 00h so that between Runtime and OCP Recovery mode the interface number remains the same regardless of static vs dynamic USB descriptors

3	bAlternateSetting	1	0x00	Alternate setting. Must be zero.
4	bNumEndpoints	1	0x00	Only the control pipe is used (EP0).
5	bInterfaceClass	1	0xEF	Miscellaneous Class Code
6	bInterfaceSubClass	1	0x08	OCP Secure Firmware Recovery SubClass assigned by USB-IF see https://www.usb.org/defined-class-codes
7	bInterfaceProtocol	1	0x01	OCP Recovery USB transport version 1.x. This version is a USB Interface specification binding version and is independent from the OCP Secure Firmware Recovery Specification versioning.
8	iInterface	1	Index	Index of string descriptor for this interface pointing to "OCP Secure Firmware Recovery"

8.5.3 USB Interface Functional Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	0x0A	Size of this descriptor in bytes.
1	bDescriptorType	1	0x24	Interface-specific/Class-specific descriptor CS_INTERFACE
2	bDescriptorSubtype	1	0x01	Interface-specific subtype for OCP Recovery CP_RECOVERY_FUNCTIONAL
3	bReserved	1	0x00	Reserved
4	wMaxWrTransferSize	2	Value	Maximum write transfer size in bytes
6	wMaxRdTransferSize	2	Value	Maximum read transfer size in bytes

8	bcdOCPRecVersion	2	BCD	Numeric expression identifying the version of the OCP Secure Firmware Recovery Specification release use the major.minor only from the document title.
---	------------------	---	-----	--

8.5.4 USB Descriptors for OCP Secure Firmware Recovery Interface

A USB Device implementing OCP Secure Firmware Recovery SHALL expose at minimum a set of descriptors for both runtime and recovery operations, including:

- A USB Device Descriptor with bDeviceClass=0, bDeviceSubClass=0, and bDeviceProtocol=0 indicating a USB Composite Device where individual interfaces define their respective classes.
- At least one Configuration Descriptor defining the USB Device's operational configuration.
- Exactly one Interface Descriptor dedicated to the OCP Secure Firmware Recovery Interface with bInterfaceClass=0xEF (Miscellaneous), bInterfaceSubClass= 0x08, bInterfaceProtocol = 0x01. Values assigned by USB-IF to uniquely identify OCP Secure Firmware Recovery interfaces across all USB versions and distinguish them from other USB Device functions see <https://www.usb.org/defined-class-codes>. These codes are only allowed in the Interface descriptor and not in the device descriptor.
- Exactly one OCP_RECOVERY_FUNCTIONAL descriptor specifying recovery capabilities and transfer limitations.

8.5.5 USB-specific Error Recovery

OCP Secure Firmware Recovery over USB EP0 employs standard USB error recovery mechanisms to handle STALL conditions as defined by USB specification that also MAY be generated by OCP Secure Firmware Recovery protocol failures in Section 9.1. Comprehensive USB-specific error recovery procedures are defined to ensure high resilience in configurations where USB serves as the sole Device recovery interface, with no fallback options such as GPIO reset pins or SMBus recovery paths. The following hierarchical escalation approach shall be implemented when OCP Secure Firmware Recovery operations fail, progressing from least to most disruptive recovery methods:

1. Attempt **CLEAR_FEATURE(ENDPOINT_HALT)**: On clearing STALL, OCP Secure Firmware Recovery assumes that the Device implementation employs sufficient methods to ensure subsequent Control Transfers are handled as expected without the need for further intervention. This includes, but is not limited to, flushing internal pipelines and resetting counters when a STALL condition is cleared by issuing **CLEAR_FEATURE(ENDPOINT_HALT)** on EP0.

2. Attempt a **USB PORT RESET** or **BUS RESET** by driving both D+ and D- lines to low state (SE0) for the minimum duration specified in the USB specification (e.g., 10ms for typical USB Device implementations). The reset shall be initiated by either a USB Hub (for port reset via **SET_PORT_FEATURE**) or USB Host Controller (for bus reset) depending on the USB physical topology. The USB reset signaling is expected to be detected by dedicated USB Device hardware (e.g., USB PHY for USB 2.0 High Speed and higher). USB reset detection shall be isolated from the USB receive/transmit logic to ensure reliable reset recognition regardless of the current USB communication state.

Each of the above methods is expected to bring the OCP Secure Firmware Recovery function on the USB Device to an operational state, at minimum enabling access to OCP Recovery mandatory capabilities. The underlying mechanism for each of the attempts is implementation specific.

8.5.6 USB Device implementation notes

The USB Device **MUST** be implemented as a USB Composite Device and **MAY** expose additional interfaces beyond the mandatory recovery interface, with the guarantee that the OCP Secure Firmware Recovery Interface remains accessible and functional under all operational states including firmware corruption and error conditions. During recovery operations, the recovery interface **SHALL** remain fully responsive while other interfaces **MAY** respond with USB STALL. Host software **SHALL** handle static USB descriptors exposing multiple interfaces and gracefully manage STALL responses from non-recovery interfaces during recovery mode. USB Device **MUST** implement USB-specific Error Recovery handling as defined in this specification. For optimal OCP Recovery flow operation, **RECOMMENDED** USB Device speed is USB 2.0 High Speed or higher.

9. Recovery Interface Commands

This section describes the command defined for the recovery interface. It is described as a generic block read and write protocol with a command byte. Not all commands are required to be implemented and it is up to the RA to determine which ones are available via the PROT_CAP command. All commands and fields are specified in little-endian format.

9.1 Error Handling/Unsupported Features

There are several errors which can occur in the protocol. The PROTOCOL_ERROR field in the Device status register is used to indicate these errors. Reading the Device status register will clear the PROTOCOL_ERROR field. Note: the PROTOCOL_ERROR is a clear on read field.

- An unsupported command to a Device **MUST** set an unsupported error condition in the DEVICE_STATUS. This includes all optional commands.

- A Device which receives (write) commands with unsupported parameters (e.g. local c-image selection when the Device it is not supported) MUST generate an ‘unsupported parameter’ error in the DEVICE_STATUS registers.
- A Device which receives (write) command with an incorrect number of bytes MUST generate an ‘length write error’ error in the DEVICE_STATUS registers.
- A Device which receives (write) command with an invalid checksum (e.g. PEC) MUST set the CRC error in the DEVICE_STATUS.
- Writing to a read only command (e.g. PROT_CAP) MUST generate an ‘unsupported command’ error in the DEVICE_STATUS.

9.2 Command Summary

Recovery Capabilities Command				
Command	r/w	bytes	Description	Req
PROT_CAP cmd=34	ro	15	Recovery protocol magic string Byte 0-7: Magic string “OCP RECV” in ASCII code - “4f 43 50 20 52 45 43 56” Recovery protocol version Byte 8: Major version number = 0x1 Byte 9: Minor version number = 0x1 Recovery protocol capabilities Byte 10-11: Agent capabilities BIT 0: Identification (DEVICE_ID structure) BIT 1: Forced Recovery (From RESET) BIT 2: Mgmt Reset (From RESET) BIT 3: Device Reset (From RESET) BIT 4: Device Status (DEVICE_STATUS) BIT 5: Recovery memory access (INDIRECT_CTRL) BIT 6: Local C-image support BIT 7: Push C-image support BIT 8: Interface isolation BIT 9: Hardware status Bit 10: Vendor command Bit 11: Flashless boot (From RESET) Bit 12: FIFO CMS support (INDIRECT_FIFO_CTRL)	Y

			Bit 13-15: Reserved Byte 12 (0-255): The total number of component memory space (CMS) regions a Device supports. This number includes any logging, code and vendor defined regions Describes the maximum amount of time an operation can take. A Device SHOULD not take more than 100 ms to respond to an operation. Byte 13: Maximum Response Time 0-255: Maximum response time in 2^x microseconds (us) Byte 14: Heartbeat Period 0-255: Heartbeat period:n 2^x microseconds (us) - 0 indicates not supported	
--	--	--	--	--

Mandatory capabilities are:

- DEVICE_ID
- DEVICE_STATUS
- Local C-image OR Push C-Image
- INDIRECT_CTRL if Push C-Image

The Device identifier is used to identify the type of Device. This DEVICE_ID command is designed to retrieve data to construct the Device identifier record per the DMTF Firmware update standard.

Command	r/w	bytes	Description	Req
ID			Device Identification	
DEVICE_ID cmd=35	ro	24- 255	Number of bytes available for each of the following IDs: Byte 0: Initial Descriptor Type:- Based on table 8 from [DMTF PLDM FM] 0x00: PCI Vendor 0x1: IANA 0x2: UUID 0x3: PnP Vendor 0x4: ACPI Vendor 0x5: IANA Enterprise Type 0x6-0xFE: Reserved 0xFF: NVMe-MI Byte 1: Vendor Specific String Length 0-0xFF: total length of Vendor Specific String . 0 indicates not supported	Y

Date: Dec 19, 2025

This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

			For PCI Type: Byte 2-3: PCI Vendor ID Byte 4-5: PCI DeviceID Byte 6-7: PCI Subsystem Vendor ID Byte 8-9: PCI Subsystem ID Byte 10: PCI Revision ID Byte 11-23: 0x0 (PAD) For UUID type: Byte 2-17: UUID assigned to the Device Byte 18-23: 0x0 (PAD) For IANA type: Byte 2-5: IANA Enterprise ID Byte 6-17: ACPI Product Identifier Byte 18-23: 0x0 (PAD) For PnP type: Byte 2-4: PnP Vendor Identifier Byte 5-8: PnP Product Identifier Byte 9-23: 0x0 (PAD) For ACPI type: Byte 2-5: ACPI Vendor Identifier Byte 6-8: Vendor Product Identifier Byte 9-23: 0x0 (PAD) For NVME-mi Byte 2-3 Vendor ID Byte 4-23: Device Serial Number Vendor Specific String Byte 24-254: ASCII encoded string	
--	--	--	---	--

Command	r/w	bytes	Description	Req
STATUS			Device Status - Accumulated Device status	
DEVICE_STATUS cmd=36	ro	7-255	Byte 0: Device status 0x0: Status Pending (Recover Reason Code not populated) 0x1: Device healthy (Recover Reason Code not populated) 0x2: Device Error (“soft” error or other error state) - (Recover Reason Code not populated)	Y

			0x3: Recovery mode - ready to accept recovery image - (Recover Reason Code populated) 0x4: Recovery Pending (waiting for activation) - (Recover Reason Code populated) 0x5: Running Recovery Image (Recover Reason Code not populated) 0x6-0xD: Reserved 0xE: Boot Failure (Recover Reason Code populated) 0xF: Fatal Error (Recover Reason Code not populated) 0x10-FF: Reserved Byte 1: Protocol Error (Clear on Read) 0x0: No Protocol Error 0x1: Unsupported/Write Command - command is not support or a write to a RO command 0x2: Unsupported Parameter 0x3: Length write error (length of write command is incorrect) 0x4: CRC Error (if supported) 0x5-0xFE: Reserved 0xFF: General Protocol Error - catch all unclassified errors Byte 2-3: Recovery Reason Codes - See table 3 Byte 4-5: Heartbeat 0-4095 - Incrementing number (counter wraps) Byte 6: Vendor Status Length 0-248: Length in bytes of just VENDOR_STATUS. Zero indicates no vendor status and zero additional bytes. Byte 7-254: Vendor Status (if vendor length is non-zero) Vendor defined status message	
--	--	--	---	--

There are various conditions where the reason code is populated. The goal is to describe why the Device failed to boot or entered recovery mode. The following table describes the source of the recovery reason code in different Device status/states.

Table 2 - Recovery Reason Code Population			
Device Status/State	Valid	Source of Recovery Reason Code	Notes
Status Pending	N	None	Device is booting or has not yet populated the reason code
Device healthy	N	None	

Date: Dec 19, 2025

This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

Device Error	N	None	Device errors may be recoverable but the Device was unable to report a recovery reason code.
Recovery Mode	Y	Previous boot	Device has entered recovery based on forced recovery, flashless boot mode, or error in the previous boot.
Recovery Pending	Y	Previous boot	Device has entered recovery based on forced recovery, or error in the previous boot.
Boot Failure	Y	Current Boot	Device current boot is halted. The reason is defined in the recovery reason code.
Fatal Error	N	None	Fatal errors are not recoverable via OCP Recovery flow.

Recovery reason codes are defined in the following table. The RCV column indicates if the Device can be recovered based on error code (Y=yes, N=no, M=possible).

Table 3 - Recovery Reason Codes

Code	Description	RCV
0x0	No Boot Failure detected (BFNF)	N
0x1	Generic hardware error (BFGHWE)	N
0x2	Generic hardware soft error (BFGSE) — soft error may be recoverable	M
0x3	Self-test failure (BFSTF) — e.g., RSA self-test failure or FIPS self-test failure	M
0x4	Corrupted/missing critical data (BFCD)	M
0x5	Missing/corrupt key manifest (BFKMMC)	Y
0x6	Authentication Failure on key manifest (BFKMAF)	Y
0x7	Anti-rollback failure on key manifest (BFKIAR)	Y
0x8	Missing/corrupt boot loader (first mutable code) firmware image (BFFIMC)	Y
0x9	Authentication failure on boot loader (1st mutable code) firmware image (BFFIAF)	Y
0xA	Anti-rollback failure boot loader (1st mutable code) firmware image (BFFIAR)	Y
0xB	Missing/corrupt main/management firmware image (BFMFMC)	Y
0xC	Authentication Failure main/management firmware image (BFMFAF)	Y
0xD	Anti-rollback Failure main/management firmware image (BFMFAR)	Y
0xE	Missing/corrupt recovery firmware (BFRFMC)	Y
0xF	Authentication Failure recovery firmware (BFRFAF)	Y
0x10	Anti-rollback Failure on recovery firmware (BFRFAR)	Y
0x11	Forced Recovery (FR)	Y
0x12	Flashless/Streaming Boot (FSB)	NA
0x13 – 7F	Reserved	NA
0x80 - FF	Vendor Unique Boot Failure Codes	NA
0x0100-0xFFFF	Reserved	NA

Command	r/w	bytes	Description	Req
DEVICE_RESET			Reset Control - combinations Reset, Recovery and Activate are possible	

RESET cmd=37	rw	3	Reset control - For Devices which support reset, this register will reset the Device or management entity. Byte 0: Device Reset Control (Write 1, Device Clears) 0x0: No reset 0x1: Reset Device (PCIe® Fundamental Reset or equivalent. This is likely bus disruptive) 0x2: Reset Management. This reset will reset the management subsystem. If supported, this reset MUST not be bus disruptive (cause re-enumeration) 0x3-FF: Reserved Byte 1: Forced Recovery 0x0 - No forced recovery 01-D - Reserved 0xE - Enter flashless boot mode on next platform reset 0xF - Enter recovery mode on next platform reset 0x10-FF: Reserved Byte 2: Interface Control 0x0: Disable Interface mastering 0x1: Enable Interface mastering	N
-----------------	----	---	---	---

Command	r/w	bytes	Description	Req
RECOVERY				
RECOVERY_CTRL cmd=38	rw	3	Recovery configuration/ctrl Selects the memory region address used for recovery. This region must be a code region. Byte 0: Component Memory Space (CMS) 0-255: Selects a component memory space where the recovery image is. 0 is the default Byte 1: Recovery Image Selection 0x0: No operation 0x1: Use Recovery Image from memory window (CMS) 0x2: Use Recovery Image stored on Device (C-image) 0x3-FF: reserved Byte 2: Activate Recovery Image (Write 1, Device Clears) 0x0 - do not activate recovery image - after activation Device will report this code. 0xF - Activate recovery image	Y

			0x10-FF-reserved	
RECOVERY_STATUS cmd=39	ro	2	Recovery status: Recovery Debug status of the Device Byte 0: Device recovery status BIT 0-3: 0x0: Not in recovery mode 0x1: Awaiting recovery image 0x2: Booting recovery image 0x3: Recovery successful 0xc: Recovery failed 0xd: Recovery image authentication error 0xe: Error entering Recovery mode (might be administratively disabled) 0xf: Invalid component address space BIT 4-7: Recovery image index Byte 1: Vendor specific status Vendor Defined	Y

Example Device Reset commands (starting with byte 0):

- Write: 0x02, 0x0F, 0x00 - Device will perform management reset AND enter forces recovery AND disables interface mastering
- Write: 0x00, 0x0F, 0x00 - Device will enter recovery on next platform reset AND disables interface mastering

Interface control is used to enable target initiated transactions (e.g., Bus mastering in SMBus). The Device must power-on with mastering disabled and the bus configuration solely managed by the RTRec. Otherwise the DEVICE_RESET command is subject to denial-of-service attacks by the Device components outside of the RTRec (e.g., management controller).

Command	r/w	bytes	Description	Req
HW_STATUS				
HW_STATUS cmd=40	ro	4-255	Byte 0: HW Status (bit mask active high) BIT 0: Device temperature is critical (may need reset to clear) BIT 1: Hardware Soft Error (may need reset to clear) BIT 2: Hardware Fatal Error BIT 3-7: Reserved Byte 1: Vendor HW Status (bit mask active high) BIT 0-7: Vendor Specific	N

Date: Dec 19, 2025

This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

			Byte 2: Composite temperature (CTemp) - Current temperature of Device in degrees Celsius: Compatible with NVMe-MI command code 0 offset 3. 0x00-0x7e: 0 to 126 C 0x7f: 127 C or higher 0x80: no temperature data, or data is older than 5 seconds 0x81: temperature sensor failure 0x82-0x83: reserved 0xc4: -60 C or lower 0xc5-0xff: -59 to -1 C (in two's complement) Byte 3: Vendor Specific Hardware Status length (bytes) 0-251: Length in bytes of Vendor Specific Hardware Status. Vendor Specific Hardware Status Byte 4-255	
--	--	--	--	--

Command	r/w	bytes	Description	Req
INDIRECT			This is the interface to memory regions within the Device	
INDIRECT_CTRL cmd=41	rw	6	Indirect memory access configuration. This register selects a region within the Device. Read/write access is through address spaces. Each space represents a location in memory which is described by the config register. The INDIRECT_OFFSET can be used to access certain offsets within an address space. Byte 0: Component Memory Space (CMS) 0-255 - Address region within a Device. Indirect memory configuration: Byte 1: Reserved Byte 2-5: Indirect memory offset (IMO) Writes and reads via INDIRECT_DATA will auto-increment this offset by the number of bytes written/read. The offset must be 4-byte aligned (e.g., the lower 2-bit are always zero). Writes and reads that are not 4-byte multiples will increment to the next 4-byte offset. Note: IMO can be read to determine the number of bytes read or written since last initialization (write to this register).	N

INDIRECT_STATUS cmd=42	ro	6	Byte 0: STATUS (bit mask) - (clear on read) BIT 0: Overflow CMS wrapped BIT 1: Read Only Error - write to a RO (log area) BIT 2: ACK from Device in a polling address space Note: The RA MAY poll with a timeout. The Device MUST respond within Maximum Responses Time reported in the PROT_ID command. If the timeout expires, then the RA must read RECOVERY_STATUS BIT 3: CMS Polling Error BIT 4: Write Only Error - read from a WO area BIT 5-7: Reserved Byte 1: Type of region: 0bP000: Code space for recovery. (read/write) 0bP001: Log uses the defined debug format (read only) 0bP100: Vendor Defined Region (write only) 0bP101: Vendor Defined Region (read/write) 0xP110: Vendor Defined Region (read only) 0bX111: Unsupported Region (address space out of range) If P is set, polling is required for this region. That is, the ACK must be set before the next indirect operation Byte 2-5: INDIRECT SIZE - size of memory window specified (by CMS in the INDIRECT_CTRL component and type) in 4B units	N
INDIRECT_DATA cmd=43	rw	1-N	Indirect memory access to address space configured in INDIRECT_CTRL at offset specified in INDIRECT_OFFSET. Note: The length of the transfer does not need to be 4-byte aligned, but the IMO will be auto-incremented to the next 4-byte offset, so data size should be 4-byte aligned for contiguous access. N in single command is limited to: - 252 bytes for SMBUS - max read data length and max write data length advertised via GETMRL and GETMWL - wMaxRdTransferSize or wMaxWrTransferSize in bytes for USB	N

Command	r/w	bytes	Description	Req
VENDOR			Vendor defined command	

VENDOR cmd=44	rw	1-N	Vendor Defined N in single command is limited to: - 255 bytes for SMBUS - max read data length and max write data length advertised via GETMRL and GETMWL for I3C - wMaxRdTransferSize or wMaxWrTransferSize in bytes for USB	N
------------------	----	-----	---	---

Command	r/w	bytes	Description	Req
INDIRECT_FIFO			This is the interface to memory regions within the Device that operate in a FIFO model.	
INDIRECT_FIFO_CTL cmd=45	rw	6	Indirect FIFO memory access configuration. This register selects a region within the Device. Read/write access is through address spaces. Each space represents a FIFO. Byte 0: Component Memory Space (CMS) 0-255 - Address region within a Device. Indirect memory configuration: Byte 1: Reset (Write 1, Device Clears) 0x0: idle 0x1: reset Write Index and Read Index to initial value causing FIFO to be empty. 0x2 to 0xFF: reserved Byte 2-5 Image Size Size of the image to be loaded in 4B units.	N
INDIRECT_FIFO_STATUS cmd=46	ro	20	Byte 0: Status BIT 0: if set, FIFO is empty BIT 1: if set, FIFO is full Byte 1: Type of region 0b000: Code space for recovery. (Write Only) 0b001: Log uses the defined debug format (Read Only) 0b100: Vendor Defined Region (Write Only) 0b101: Vendor Defined Region (Read Only) 0b111: Unsupported Region (Address Space Out of Range) Byte 2-3: Reserved Byte 4-7: Write Index	N

			For Write Only Regions Offset incremented for each access by the Recovery Agent in 4B units For Read Only Regions Offset incremented for each access by the Device in 4B units Byte 8-11: Read Index For Write Only Regions Offset incremented for each access by the Device in 4B units For Read Only Regions Offset incremented for each access by the Recovery Agent in 4B units Byte 12-15: Indirect FIFO size Size of memory window specified in 4B units Byte 16-19: Max transfer size Max size of the data payload in each read/write to INDIRECT_FIFO_DATA in 4B units Max transfer size MAY be different between read and write direction see chapter 8 for I3C and USB details. For unidirectional regions read only and write only provide the read or write length respectively to region definition. For read/write regions provide the lowest common denominator. RECOMMENDED for new region definition is to add support for unidirectional regions only.	
INDIRECT_FIFO_DATA cmd=47	rw	1-N	Indirect memory access to address space configured in INDIRECT_FIFO_CTRL at the Head Pointer offset. Note: The length of the transfer does not need to be 4-byte aligned, but the Head Pointer will be auto-incremented to the next 4-byte offset, so data size should be 4-byte aligned for contiguous access. N in single command is limited to: - 252 bytes for SMBUS - max read data length and max write data length advertised via GETMRL and GETMWL for I3C - wMaxRdTransferSize or wMaxWrTransferSize in bytes for USB	N

10. Protocol Conformance Checklist/Statement

The following table is used to describe a Device compliance to the recovery protocol.

Date: Dec 19, 2025

This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

Table 4 - Recovery Protocol Compliance Statement

Feature	Description	Compliance
PROT_CAP mandatory	Mandatory capabilities:	DEVICE_ID, DEVICE_STATUS, Local or Push C-image, INDIRECT if push C-image
PROT_CAP optional	Optional capabilities:	forced recovery, mgmt_reset, hardware status, recovery_memory access, heartbeat, flashless boot
PROT_CAP	Response time	2^x microseconds
PROT_CAP	Protocol version	{0x1, 0x0}
PROT_CAP	Heartbeat support	Yes or No
PROT_CAP	Heartbeat period values	2^x microseconds
PROT_CAP	Recovery image type	Push, local, both
PROT_CAP	CMS Support	Yes or No
PROT_CAP	Device Bus Isolation	Yes or No
DEVICE_ID	Supported Device ID format	PCI Vendor, IANA, UUID, PnP Vendor, ACPI Vendor, or NVMe-MI
DEVICE_ID	Vendor specific string	Value and length
DEVICE_STATUS	Recovery Reasons codes	Enumerate supported reasons codes from table 3
DEVICE_STATUS	Vendor Status	Supported? Length and description
DEVICE_RESET	Management reset support	Yes or No
DEVICE_RESET	Device reset support	Yes or No
DEVICE_RESET	Forced recovery support	Yes or No
RECOVERY_CTRL	Recovery Status support	Yes or No
HW_STATUS	HW status support	Yes or No

HW_STATUS	Vendor HW Status	Yes or No, description if yes
HW_STATUS	Composite temperature support	Yes or No
INDIRECT	Indirect Access command support	Yes or No
INDIRECT	Number and size of CMS code spaces	0-255, 0 – 4GB
INDIRECT	Number of CMS log spaces	0-255, 0-4G
INDIRECT	Number of Vendor CMS code spaces	0-255, 0-4G
INDIRECT	Number of Vendor CMS log spaces	0-255, 0-4G
INDIRECT	Is polling required for any CMS	Yes or No
VENDOR	Is vendor defined command supported	Yes or No, description if yes
SMBUS	Speed classes supported	100K, 400K, 1M
SMBUS	PEC support	Yes or No
SMBUS	ARP support	Yes or No
SMBUS	Fixed address support	Yes or No, is this Device configurable?
INDIRECT_FIFO	Indirect FIFO	Does the CMS NACK buffer overflows?

The following is a list of tests which SHOULD be performed to demonstrate compliance to this protocol. The list is not exhaustive, but is designed to give implementer guidance from the authors as to the important or non-obvious parts of the protocol.

Table 5 - Recovery Protocol Error/Test		
Test	Description	Compliance
Unsupported commands	Read and write using unsupported commands	Verify protocol error and COR behavior
Read Only	Write to read only commands	Verify protocol error and COR behavior
Write Error	Write a command with an incorrect number of bytes	Verify protocol error and COR behavior

Date: Dec 19, 2025

This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

Write PEC error	If supported write using an incorrect PEC	Verify correct SMBus behavior and Verify protocol error
Device Status	Read Device status while Device is not ready	The Device must report pending in the Device status register.

11. Glossary and Abbreviations

See [Glossary and Abbreviations](#)

12. Relevant standards, guidelines, and documents

- [1] NIST Special Publication 800-155 (DRAFT), [BIOS Integrity Measurement Guidelines](#)
- [2] NIST Special Publication 800-193, [Platform Firmware Resiliency Guidelines](#)
- [3] [Open Compute Project, Project Cerberus Firmware Update Specification](#)
- [4] [SMBus 3.1 Specification](#)
- [5] [OCP Attestation of System Components v1.0 Requirements and Recommendations](#)
- [6] [NVM-Express-Management-Interface-1.2a](#)
- [7] DMTF PLDM for Firmware Update [DSP0267 1.1.0.pdf \(dmtof.org\)](#)
- [8] [OCP NVMe Cloud SSD Specification \(v1.0\)](#)
- [9] USB 2.0 Specification <https://www.usb.org/document-library/usb-20-specification>

13. License

OCP encourages participants to share their proposals, specifications and designs with the community. This is to promote openness and encourage continuous and open feedback. It is important to remember that by providing feedback for any such documents, whether in written or verbal form, that the contributor or the contributor's organization grants OCP and its members irrevocable right to use this feedback for any purpose without any further obligation.

It is acknowledged that any such documentation and any ancillary materials that are provided to OCP in connection with this document, including without limitation any white papers, articles, photographs, studies, diagrams, contact information (together, "Materials") are made available under the Creative Commons Attribution-ShareAlike 4.0 International License found here: <https://creativecommons.org/licenses/by-sa/4.0/>, or any later version, and without limiting the foregoing, OCP may make the Materials available under such terms.

As a contributor to this document, all members represent that they have the authority to grant the rights and licenses herein. They further represent and warrant that the

Date: Dec 19, 2025

This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](#).

Materials do not and will not violate the copyrights or misappropriate the trade secret rights of any third party, including without limitation rights in intellectual property. The contributor(s) also represent that, to the extent the Materials include materials protected by copyright or trade secret rights that are owned or created by any third-party, they have obtained permission for its use consistent with the foregoing. They will provide OCP evidence of such permission upon OCP's request. This document and any "Materials" are published on the respective project's wiki page and are open to the public in accordance with OCP's Bylaws and IP Policy. This can be found at <http://www.opencompute.org/participate/legal-documents/>. If you have any questions, please contact OCP.

14. About Open Compute Foundation

The Open Compute Project (OCP) brings at-scale innovations and hyperscaler best practices to all, spanning technology domains from the data center to the edge, and the technology stack from silicon, to systems, to site facilities and services. The international OCP Community is made up of organizations and people from hyperscale and tier-2 cloud data center operators, communications providers, colocation providers, diverse enterprises, and technology vendors. The OCP Foundation fosters collaboration within the Community to tackle market challenges in an array of OCP Projects that create open contributions such as specifications, designs, and more. The OCP Solution Provider Program then recognizes the application of those contributions in compliant products, solutions, and data center facilities and services with designations like OCP Inspired, OCP Accepted and OCP Ready. With the tenets of openness, impact, efficiency, scale and sustainability, the OCP engages and educates thousands of engineers every year through many events and webinars. Across many initiatives the OCP Foundation and Community are meeting the market today and shaping the future.

Learn more at: www.opencompute.org.