ARM® Management Mode Interface Specification

Document number: ARM DEN 0060A

ARM Management Mode Interface Specification
System Software on ARM

Release information

The Change History table lists the changes that are made to this document.

Table 1-1 Change history

Date	Issue	Confidentiality	Change
06 December 2016	A	Non-Confidential	First release

Contents

1 About this Document

This document describes standard SMC functions to be used for software invocation of Management Mode (MM) services.

1.1 References

This document refers to the following documents and links.

Reference	Document name	Document number
[1]	VOLUME 4: Platform Initialization Specification. Management Mode Core Interface	UEFI Platform Initialization Specification Version 1.5
[2]	Unified Extensible Firmware Interface Specification	UEFI version 2.6
[3]	SMC Calling Conventions specification	ARM DEN 0028
[4]	Advanced Configuration and Power Interface Specification	ACPI 6.1
[5]	ARM Trusted Firmware repository	https://github.com/ARM-software/arm-trusted-firmware
[6]	Tianocore EDK2 repository	https://github.com/tianocore/edk2
[7]	ARM Architecture Reference Manual ARMv8, for ARMv8-A architecture profile	ARM DDI 0487A

1.2 Terms and abbreviations

This document uses the following terms and abbreviations.

Term	Description
DXE	Driver Execution Environment
EDK	EFI Developer Kit
MM	Management mode. A mode of execution agnostic to the operating system used to provide platform management firmware. For more details see [1] .
Non-secure state	The ARM Execution state that restricts access to only the Non-secure system resources such as memory, peripherals, and System registers
Normal World	The execution environment when the core is in the Non-secure state
OS	Operating System
PEI	Pre-EFI Initialization
PI	Platform Initialization
PIWG	Platform Initialization Working Group
Secure state	The ARM Execution state that enables access to the Secure and Non-secure systems resources, such as memory, peripherals, and System registers
Secure world	The execution environment when the core is in the Secure state
UEFI	Unified Extensible Firmware Interface. For more details, see *Unified Extensible Firmware Interface Specification .*

1.3 Feedback

ARM welcomes feedback on its documentation.

1.3.1 Feedback on this manual

If you have comments on the content of this manual, send an e-mail to errata@arm.com. Give:

- The title.
- The document and version number, ARM DEN 0060A.
- The page numbers to which your comments apply.
- A concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

2 Introduction

Management Mode (MM) provides an environment for implementing OS agnostic services (MM services) like RAS error handling, secure variable storage, and firmware updates in system firmware. The services can be invoked synchronously and asynchronously.

An example of a synchronous invocation could be an access to secure storage, from the OS or hypervisor, to read or write data. This document describes interfaces for invoking MM services synchronously.

An example of asynchronous invocation could be a RAS event signaled as an exception. A description of how MM services can be invoked asynchronously is beyond the scope of this specification.

2.1 Security state considerations

ARMv8-A architecture describes multiple *Exception levels* (ELs) across the Secure and Non-secure Security states. If EL3 is not implemented or the services are being virtualized in EL2, MM services can be implemented either in the Secure world or in the Normal world in EL2..

This document describes the MM interface and implementation considerations for invoking MM services from the Normal world. It does not preclude access to MM services that are implemented in the Secure world from other software components that are also in the Secure world.

The PIWG in the UEFI forum maintains [1], which describes an MM environment in which MM services can be implemented. An implementation of this specification and MM services exists in the *Tianocore EDK2 repository*.

The *ARM Trusted Firmware repository* provides a reference implementation of secure system firmware for ARMv8-A platforms. One approach to implementing MM services on ARMv8-A platforms is to:

1. Implement an ARM Trusted Firmware managed MM environment that is based on EDK2 sources in the Secure world.
2. Use the MM interfaces that are described in this document from the Normal world, for example, UEFI DXE drivers, to access this environment and services.

2.2 Execution state considerations

The ARMv8 architecture defines two Execution states, AArch32 and AArch64 (See [7]). It is possible for the Execution state to differ between the caller and the implementation. MM services are only supported if the implementation uses an Execution state that is higher than or equal to the Execution state of the caller. Table 2 details which combinations are permitted.

Table 2 Permitted MM implementations

Exception level of caller Execution state	MM implementation level Execution state	Validity
AArch32	AArch32	Permitted
AArch32	AArch64	Permitted
AArch64	AArch64	Permitted
AArch64	AArch32	Not permitted

2.3 Conduits

The *SMC Calling Conventions specification* describes the SMC and HVC conduits for accessing firmware services and their availability depending on the implemented Exception levels.

If EL3 is implemented, ARM recommends that the SMC conduit is used. When MM services are accessed from the Normal world at EL1, SMC execution can be trapped by a hypervisor at EL2. This means that the SMC conduit provides the flexibility that is required to support implementations with and without a hypervisor.

If EL2 is implemented and EL3 is not implemented, the HVC conduit must be used. In this case, it is the only conduit available.

2.4 Calling conventions

The *SMC Calling Conventions specification* describes the 32-bit and 64-bit calling conventions for the SMC and HVC conduits. If the conduit is HVC, this document assumes that the 32-bit (SMC32) and 64-bit (SMC64) calling conventions are equivalent to the HVC32 and HVC64 calling conventions, respectively.

3 Interface

This specification reserves function IDs for Fast calls (See [3]), in the Standard Secure Service (See [3]) calls range, for each interface to access MM services. This section defines the function prototypes for each function ID. The function IDs specify whether one or both of the SMC32 and SMC64 calling conventions can be used to invoke the corresponding interface.

3.1 MM_VERSION

Description

Returns the version of the MM implementation

Parameters

Declaration	Value
uint32 Function ID	• 0x8400 0040

Return

Declaration	Value
int32	On success, the format of the value is as follows: • Bit [31]: Must be 0 • Bits [30:16] Major Version: Must be 1 for this revision of MM • Bits [15:0] Minor Version: Must be 0 for this revision of MM
	On error, the format of the value is as follows: • NOT_SUPPORTED: MM is not supported or not available for the client See Section 3.3 for integer values that are associated with each return code.

3.1.1 Usage

This function returns the version of the MM interface implementation. Each implemented MM interface must support this call and return its implementation version. For this revision of the MM interface, the major version is 1 and the minor version is 0.

The version number is a 31-bit unsigned integer, with the upper 15 bits denoting the major revision, and the lower 16 bits denoting the minor revision. The following rules apply to the version numbering:

- Different major revision values indicate possibly incompatible functions.
- For two revisions, A and B, for which the major revision values are identical, if the minor revision value of revision B is greater than the minor revision value of revision A, then every function in revision A must work in a compatible way with revision B. However, it is possible for revision B to have a higher function count than revision A.

3.1.2 Implementation responsibilities

If this function returns a valid version number, all the functions that are described in this specification must be implemented, unless it is explicitly stated that a function is optional.

3.2 MM_COMMUNICATE

Description	
Invokes an MM service	

Parameters	
Declaration	**Value**
uint32 Function ID	• 0x8400 0041 • 0xC400 0041
uint32/uint64 Cookie	Reserved for future use. Must be zero.
uint32/uint64 comm_buffer_address	See section 3.2.2 For the SMC64 version, this parameter is a 64-bit *Physical Address* (PA) or *Intermediate Physical Address* (IPA). For the SMC32 version, this parameter is a 32-bit PA or IPA.
unit32/unit64 comm_size_address	See section 3.2.2 For the SMC64 version, this parameter is a 64-bit PA or IPA. For the SMC32 version, this parameter is a 32-bit PA or IPA.

Return	
Declaration	**Value**
int32	On success, the format of the value is as follows: • SUCCESS
	On error, the format of the value is as follows: • NOT_SUPPORTED • INVALID_PARAMETER • NO_MEMORY • DENIED See Section 3.3 for integer values that are associated with each return code.

3.2.1 Usage

Calling this function invokes an MM service that is implemented in EL2 or the Secure world. If multiple services are implemented, the service that is targeted by this call must be identified through an IMPLEMENTATION DEFINED mechanism. For example, the communication buffer that is addressed by the **`comm_buffer_address`** parameter could contain data to identify and invoke an MM service.

3.2.2 Parameters

In addition to the function ID and the cookie field, the function takes the following parameters:

- **`comm_buffer_address`** is a PA or an IPA to a communication buffer. The buffer must be allocated in physically contiguous memory. See Section 3.2.3 for more details about the attributes of this parameter.

- **comm_size_address** is a PA or an IPA that holds the size of the communication buffer being passed in.

  This parameter is optional and can be omitted by passing a zero. ARM does not recommend using it since this might require the implementation to create a separate memory mapping for the parameter. ARM recommends storing the buffer size in the buffer itself. An example of this approach is provided in Section 4.

  If this parameter is used, see Section 3.2.3 for more details about its attributes.

3.2.3 Communication buffer attributes

The communication buffer is a subset of a memory region that is allocated specifically for MM communication. It is accessed from separate translation regimes by the caller and the implementation of the MM_COMMUNICATE function. To avoid a mismatch of memory attributes, both the caller and the implementation must use the attributes that are described in this section to map this memory region in their respective translation regimes. The same attributes must be used for the region of memory that is referenced by the **comm_size_address** parameter, if it is used.

- The memory region must be mapped with the following memory region attributes and data access permissions:
  - Normal Write-Back Cacheable.
  - Non-transient Read-Allocate.
  - Non-transient Write-Allocate.
  - Inner Shareable.
  - Read-Write.
- The memory region must be mapped with Execute-Never instruction access permissions in the implementation translation regime. ARM recommends that the same is done in the caller translation regime.
- The base address of the memory region must be aligned to the maximum translation granule size that is specified in the ID_AA64MMFR0_EL1 System register. See *ARM Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*.
- The size of the memory region must be a multiple of the size of the maximum translation granule size that is specified in the ID_AA64MMFR0_EL1 System register.

3.2.4 Caller responsibilities

The caller must be able to handle the following potential return error codes, which are described in Section 3.3:

- NOT_SUPPORTED is returned when MM services that are capable of being invoked synchronously are not present.
- INVALID_PARAMETER is returned if **comm_buffer_address** is 0.
- DENIED is returned when the address passed is known to be in an address range that must not be accessed by MM services.

  NO_MEMORY is returned when MM services do not have sufficient memory resources to deal with the size of the incoming buffer. If **comm_size_address** is passed, then the address pointed to must be updated with the size of the overall communication buffer that the implementation can tolerate.

3.3 Return codes

Table 3 defines the values for return codes that are used with the interface functions. The error return type is a 32-bit signed integer. Zero and positive values denote success and negative values indicate error.

Table 3 Return codes and values

Name	Value
SUCCESS	0
NOT_SUPPORTED	-1
INVALID_PARAMETER	-2
DENIED	-3
NO_MEMORY	-5

4 Interoperability with PI MM specification

[1] describes protocols that export interfaces for communicating with MM services. The PEI phase describes the EFI_PEI_MM_COMMUNICATION_PPI protocol which exports the EFI_PEI_MM_COMMUNICATE interface. The DXE phase describes the EFI_MM_COMMUNICATION_PROTOCOL which exports the EFI_MM_COMMUNICATE interface. Both interfaces are similar semantically, and are responsible for:

1. Preparing input parameters for consumption by the MM environment. The parameters identify the MM service and provide arguments for the service.
2. Invoking the MM environment by effecting a mode transition, since the MM environment executes in a different mode to the UEFI and the OS.

The CommBuffer and CommSize parameters of these interfaces map to the comm_buffer_address and comm_size_address parameters of the MM_COMMUNICATE call that is described in Section 3.2. The implementation of the function must use an HVC or SMC conduit as determined from the ARM_BOOT_ARCH field in the ACPI FADT. See *Advanced Configuration and Power Interface Specification*. Table 4 shows the mapping of parameters and error return codes between the PI communication interfaces described in this section and the MM_COMMUNICATE SMC.

Table 4 Parameter and error return code mapping

EFI_MM_COMMUNICATE function	**MM_COMMUNICATE SMC**
CommSize	comm_size_address
CommBuffer	comm_buffer_address
EFI_SUCCESS	SUCCESS
EFI_BAD_BUFFER_SIZE	NO_MEMORY
EFI_INVALID_PARAMETER	INVALID_PARAMETER
EFI_ACCESS_DENIED	DENIED

The comm_buffer_address must start with an EFI_MM_COMMUNICATE_HEADER. See [1] for details. The MessageLength field of the header must be populated and correct prior to the call.

ARM recommends not using the optional comm_size_address parameter, or, if used, placing the pointer in the same page as the comm_buffer_address parameter, so that the implementation does not have to create an additional memory mapping. If the parameter is passed, the address that is specified must contain the full size of the communication buffer, including the EFI_MM_COMMUNICATE_HEADER.

If NO_MEMORY is returned when comm_size_address is passed, then the address pointed to is updated with the size of the communication buffer that the implementation can accept. This includes the size of the header. The header contains native fields where the size of the field depends on the Execution state of the caller, AArch32 or AArch64. The size of the header that is considered when updating the address must be relative to the Execution state of the caller.