NO LICENSE (EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE) TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT.

INTEL DISCLAIMS ALL EXPRESS AND IMPLIED WARRANTIES, INCLUDING WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT, AS WELL AS ANY WARRANTY ARISING FROM COURSE OF PERFORMANCE, COURSE OF DEALING, OR USAGE IN TRADE.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products and services described may contain defects or errors known as errata, which may cause deviations from published specifications. Current characterized errata are available on request.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

Copies of documents that have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting www.intel.com/design/literature.htm.

Intel, the Intel logo, and Xeon are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others

Copyright© 2021, Intel Corporation. All Rights Reserved.
Contents

1 Introduction ................................................................................................................................................. 1
  1.1 Background ...........................................................................................................................................................1
  1.2 Overview ................................................................................................................................................................1
    1.2.1 MM Code Injection ....................................................................................................................................2
    1.2.2 MM Driver Update .....................................................................................................................................2
    1.2.3 MM Telemetry .....................................................................................................................................2
  1.3 Terminology .........................................................................................................................................................2
  1.4 References .............................................................................................................................................................2

2 ACPI DSM Interface ........................................................................................................................................ 3
  2.1 ACPI Interface Overview ..................................................................................................................................3
  2.2 MM Runtime Update .........................................................................................................................................5
    2.2.1 _DSM Method ......................................................................................................................................5
    2.2.2 Return Status Values ........................................................................................................................ 5
    2.2.3 Function Index 1 – Query Update Capability .................................................................................6
    2.2.4 Function Index 2 – Query Communication Buffer Info ..................................................................8
    2.2.5 Function Index 3 – Start MM Runtime Update ..............................................................................8
  2.3 MM Telemetry ................................................................................................................................................... 10
    2.3.1 _DSM Method ...................................................................................................................................10
    2.3.2 Return Status Values ....................................................................................................................10
    2.3.3 Function Index 1 – Set MM Log Level ............................................................................................11
    2.3.4 Function Index 2 – Get MM Log Level ..........................................................................................12
    2.3.5 Function Index 3 – Get MM Log Data Info ..................................................................................12

A Appendix (OS Agent Sample Algorithm) ............................................................................................ 15
  A.1 Telemetry Buffer Data Retrieval ...........................................................................................................15
  A.2 Delta Telemetry Data Retrieval ........................................................................................................15
Figures

Figure 1-1. MM Runtime Update System ................................................................................................................. 1
Figure 2-1. MM Runtime Update Flow ....................................................................................................................... 4

Tables

Table 1-1. Terminology.................................................................................................................................................... 2
Table 1-2. Reference Documents .............................................................................................................................. 2
Table 2-1. SMRU Device Function Index .................................................................................................................. 5
Table 2-2. Return Status Values ................................................................................................................................ 6
Table 2-3. Query Update Capability – Return Package Values .............................................................................. 7
Table 2-4. Query Communication Buffer Info – Return Package Values .............................................................. 8
Table 2-5. Start MM Runtime Update – Input Package Values .............................................................................. 9
Table 2-6. Code Inject – Return Package Values ................................................................................................... 9
Table 2-7. SMTS Device Function Index ................................................................................................................ 10
Table 2-8. Set MM Log Level – Input Package Values .......................................................................................... 11
Table 2-9. Set MM Log Level – Return Package Values ....................................................................................... 11
Table 2-10. Get MM Log Level – Return Package Values .................................................................................... 12
Table 2-11. Get MM Log Data Info - Input Package Values ................................................................................ 13
Table 2-12. Get MM Log Data – Return Package Values .................................................................................... 14

Revision History

<table>
<thead>
<tr>
<th>Revision Number</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>• Initial release</td>
<td>August 2021</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Background

Certain compute systems require high service level agreements (SLAs) where fewer system reboot firmware updates are required for deploying firmware changes to address bug fixes, security updates, and to debug and root cause issues. Intel's solution is called Intel® Seamless Update. The management mode (MM), UEFI runtime services and ACPI services handle most of the system runtime functions. Intel® processor architecture supports MM through System Management Mode (SMM). Changing the MM code execution during runtime is called MM Runtime Update (MRU).

1.2 Overview

The MM Runtime Update mechanism incorporates the following features:

- MM Code Injection
- MM Driver Update
- MM Telemetry

These features are designed to be invoked from OS through ACPI Device Specific Method (DSM). The Figure 1-1 shows the high-level architecture of the MM runtime update infrastructure.

Figure 1-1. MM Runtime Update System
1.2.1 MM Code Injection

MM code injection feature provides a framework to deliver firmware image capsule to the MM that is designed to run in MM context for the given platform. The code update infrastructure allows delivering runtime firmware changes that are targeted for hardware and firmware state modifications to allow security or performance enhancements.

1.2.2 MM Driver Update

MM driver update feature provides a framework to deliver firmware image capsules to the MM that is designed to upgrade MM drivers. The driver update infrastructure allows delivering runtime firmware changes that are targeted for firmware driver changes to allow security or performance enhancements.

1.2.3 MM Telemetry

MM Telemetry feature provides a framework to retrieve log messages from MM for monitoring and root cause of issues.

1.3 Terminology

Table 1-1. Terminology

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACPI</td>
<td>Advanced Configuration and Power Interface</td>
</tr>
<tr>
<td>BIOS</td>
<td>Basic Input/Output System</td>
</tr>
<tr>
<td>DSM</td>
<td>Device Specific Method</td>
</tr>
<tr>
<td>MM</td>
<td>Management Mode</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>SLA</td>
<td>Service Level Agreement</td>
</tr>
<tr>
<td>SMM</td>
<td>System Management Mode</td>
</tr>
<tr>
<td>MRU</td>
<td>MM Runtime Update</td>
</tr>
<tr>
<td>SMRU</td>
<td>Seamless MM Runtime Update</td>
</tr>
<tr>
<td>SMTS</td>
<td>Seamless MM Telemetry Service</td>
</tr>
<tr>
<td>UEFI</td>
<td>Unified Extensible Firmware Interface</td>
</tr>
<tr>
<td>UUID</td>
<td>Universally Unique Identifier</td>
</tr>
</tbody>
</table>

1.4 References

Table 1-2. Reference Documents

<table>
<thead>
<tr>
<th>Reference</th>
<th>Document</th>
<th>Document Location</th>
</tr>
</thead>
</table>
2 ACPI DSM Interface

2.1 ACPI Interface Overview

A new ACPI namespace device is defined in this specification to facilitate the OS-BIOS communication of MM Runtime Update. Platforms that support MM Runtime Update must follow this specification to:

- Produce ACPI SMRU Device (Section 2.1) and required _DSM Method (Section 2.2.1).
- Provide an MM Communication Buffer for OS to deliver the MM Runtime Update Capsule Image to MM Runtime Update Handler.
- Provide MM Telemetry Service for OS to collect telemetry information during MM Runtime Update process.

An OS agent shall use the ACPI device DSM method defined in this specification to interact with platform BIOS to perform the MM Runtime Update. The MM Runtime Update workflow is shown in Figure 2-1. MM Runtime Update Flow Figure 2-1.
Figure 2-1. MM Runtime Update Flow

System Init Flow

- Produce ACPI Device with DSM Method
- Reserve MM Communication Buffer for update capsule image
- Reserve MM Communication Buffer for telemetry data

Runtime Update Flow

- Enumerate ACPI device
- Query MM Runtime Update capability and revision info (DSM function)
  - MM for update capability & revision info
- Query capsule image MM Communication buffer info (DSM function)
- Copy capsule image into MM Communication buffer (Optional) Set telemetry log level (DSM function)
  - MM to set telemetry log level
- Invoke MM Runtime Update (DSM function)
  - MM to invoke MM update capsule process

- Update Result
- Get MM telemetry log data (DSM function)
  - Data Address, Size, Overflow Indicator

Note: An OS agent may repeat the "Runtime Update Flow" to deliver multiple MM Runtime Update capsule images to system firmware, or rollback a previous update in one power cycle.
2.2 MM Runtime Update

In order to handle MM Runtime Update, the OS must first be able to detect and enumerate the SMRU (Seamless MM Runtime Update) device. An SMRU device is represented by an ACPI namespace device with a _HID or _CID object value of "INTC1080".

An SMTS Device must contain the _DSM method required to program the MM Runtime Update as defined in Section 2.2.1.

Example:

```
Scope (_SB) {
  Device (SMRU) {
    Name (_HID, "INTC1080")
    Name (_STR, …)

    Method (_DSM, …) {
      …
    }
  }
}
```

2.2.1 _DSM Method

The _DSM method for an SMRU device is described in this section.

Input Parameters:

Arg0 – UUID (set to ECF9533B-4A3C-4E89-939E-C77112601C6D)
Arg1 – Revision ID (set to 1)
Arg2 – Function Index

Table 2-1. SMRU Device Function Index

<table>
<thead>
<tr>
<th>Function Index</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Standard _DSM query function</td>
</tr>
<tr>
<td>1</td>
<td>Query Update Capability (see Section 2.2.3)</td>
</tr>
<tr>
<td>2</td>
<td>Query Communication Buffer Info (see Section 2.2.4)</td>
</tr>
<tr>
<td>3</td>
<td>Start MM Runtime Update (see Section 2.2.5)</td>
</tr>
<tr>
<td>All other value</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

Arg3 – A package containing parameters for the function specified by the UUID, Revision ID and Function Index. The layout of the package for each command along with the corresponding output is illustrated in the respective Function Index description sections.

2.2.2 Return Status Values

If Function Index is not zero, the output package in this DSM may contain a Status field. The return status values are defined in next table.
Table 2-2. Return Status Values

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| Status     | 0 – Success  
             1 – Function Not Supported  
             2 – Invalid Input Parameters  
             3 – Hardware Error  
             4 – Retry Suggested  
             5 – Unknown Reason  
             6 – Function Specific Error (details in Extended Status field)  
             All other value – Reserved |
| Extended Status | Function Specific |

2.2.3 Function Index 1 – Query Update Capability

This function allows software to get the MM Runtime Update capability.

Function Input

None

Function Output

A package as described next.

```
Package {
    Status     //Integer
    Update Capability // Integer
    MM Code Injection Image Type // Buffer
    MM Firmware Version // Integer
    MM Code Injection Runtime Version // Integer
    MM Driver Update Image Type // Buffer
    MM Driver Update Runtime Version // Integer
    MM Driver Update Runtime SVN // Integer
    Platform ID // Buffer
    OEM ID // Buffer
    OEM Information // Buffer
}
```
Table 2-3. Query Update Capability – Return Package Values

<table>
<thead>
<tr>
<th>Field</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>Integer</td>
<td>Defined in Table 2-2</td>
</tr>
</tbody>
</table>
| Update Capability      | Integer                 | Multiple bits can be set to indicate the supported MM Runtime Update features. A bit is set to one to indicate the feature is supported, zero means not supported.  
Bit 0 – MM Code Injection  
Bit 1 – MM Driver Update  
Others – Reserved and must be 0. |
| MM Code Injection      | Buffer (16 Bytes)       | A buffer containing an image type GUID.  
Platform supported capsule image type. The MM code injection capsule image must have the same Update Image Type ID. |
The supported firmware version value inside an MM Runtime Update capsule image must have same value as this field. |
An injected code image must have equal or higher image version than this value. This field is always initialized to 0 in the firmware boot.  
This field is invalid and must be ignored if MM Code Injection is not supported. |
| MM Driver Update       | Buffer (16 Bytes)       | A buffer containing an image type GUID.  
Platform supported MM Driver Update image type. The MM driver update capsule image must have the same Update Image Type ID. |
| MM Driver Update       | Integer                 | The version of the SMI Driver Update runtime code.  
This field is invalid and must be ignored if MM Driver Update is not supported. |
| MM Driver Update       | Integer                 | The secure version number (SVN) of the SMI Driver Update Runtime code.  
The MM Driver Update Runtime Version of new capsule must be no less than the SVN in current BIOS.  
This field is invalid and must be ignored if MM Driver Update is not supported. |
| Platform ID            | Buffer (16 Bytes)       | A buffer containing a platform ID GUID.  
This is a platform specific GUID to specify the platform what this capsule image support. An MM driver update capsule image must have same Platform ID. |
| OEM ID                 | Buffer (16 Bytes)       | A buffer containing an OEM ID GUID.  
This is a vendor specific GUID to specify the OEM ID. An MM driver update capsule image must have a same OEM Header Type with this value. |
| OEM Information        | Buffer                  | A buffer containing the vendor specific information.  
This is a buffer object that contains the vendor specific data. OS shall check the OEM ID to determine how to interpret this OEM information data. |
2.2.4 Function Index 2 – Query Communication Buffer Info

This function allows software to get the communication buffer information of MM Runtime Update.

**Function Input**

None

**Function Output**

A package as described next.

```
Package {
    Status     // Integer
    Extended Status   // Integer
    Buffer Address Low   // Integer
    Buffer Address High   // Integer
    Buffer Size    // Integer
}
```

**Table 2-4. Query Communication Buffer Info – Return Package Values**

<table>
<thead>
<tr>
<th>Field</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>Integer</td>
<td>Defined in Table 2-2</td>
</tr>
<tr>
<td>Extended Status</td>
<td>Integer</td>
<td>Implementation specific</td>
</tr>
<tr>
<td>Buffer Address Low</td>
<td>Integer</td>
<td>Low 32-bit physical address of the communication buffer to hold an MM Runtime Update Package.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> This field is not applicable to OOB MM Update.</td>
</tr>
<tr>
<td>Buffer Address High</td>
<td>Integer</td>
<td>High 32-bit physical address of the communication buffer to hold an MM Runtime Update Package.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> This field is not applicable to OOB MM Update.</td>
</tr>
<tr>
<td>Buffer Size</td>
<td>Integer</td>
<td>Maximum size in bytes of the communication buffer.</td>
</tr>
</tbody>
</table>

2.2.5 Function Index 3 – Start MM Runtime Update

This function instructs the firmware to process the MM Runtime Update Capsule image in communication buffer.

Before invoking this function, an MM Runtime Update Capsule image must be placed in the communication buffer specified by [Section 2.2.4](#).

If there is something wrong in the MM Runtime Update, it may be rolled back to the last known good version by issuing another call of this function, with a special capsule image or an older version capsule image.
**Function Input**

```plaintext
Package {
  Action     // Integer
}
```

**Table 2-5. Start MM Runtime Update – Input Package Values**

<table>
<thead>
<tr>
<th>Field</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>Integer</td>
<td>Indicate what action is required to perform for the capsule image:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 – Stage a capsule image from communication buffer into MM and perform authentication.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 – Activate a previous staged capsule image.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 – Perform both stage and activation actions.</td>
</tr>
</tbody>
</table>

**Function Output**

A package as described next.

```plaintext
Package {
  Status     // Integer
  Extended Status     // Integer
  Authentication Time Low     // Integer
  Authentication Time High     // Integer
  Execution Time Low     // Integer
  Execution Time High     // Integer
}
```

**Table 2-6. Code Inject – Return Package Values**

<table>
<thead>
<tr>
<th>Field</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>Integer</td>
<td>Defined in Table 2-2</td>
</tr>
<tr>
<td>Extended Status</td>
<td>Integer</td>
<td>Implementation specific</td>
</tr>
<tr>
<td>Authentication Time Low</td>
<td>Integer</td>
<td>Low 32-bit value of image authentication time in nanosecond.</td>
</tr>
<tr>
<td>Authentication Time High</td>
<td>Integer</td>
<td>High 32-bit value of image authentication time in nanosecond.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A value of zero in Authentication Time Low and Authentication Time High means authentication time is not reported.</td>
</tr>
<tr>
<td>Execution Time Low</td>
<td>Integer</td>
<td>Low 32-bit value of image execution time in nanosecond.</td>
</tr>
<tr>
<td>Execution Time High</td>
<td>Integer</td>
<td>High 32-bit value of image execution time in nanosecond.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A value of zero in Execution Time Low and Execution Time High means execution time is not reported.</td>
</tr>
</tbody>
</table>

Note that this function only measures the cryptographic authentication and image execution time of the MM runtime update image. The time of processor operating mode switching (entering/exiting MM) and SMI handler dispatching is excluded.
2.3 MM Telemetry

The OS must be able to detect and enumerate the SMTS (Seamless MM Telemetry Service) device to an SMTS device is represented by an ACPI namespace device with a _HID or _CID object value of “INTC1081”.

An SMTS Device must contain the _DSM method required to program the MM Runtime Update as defined in Section 2.3.1.

Example:

```plaintext
Scope (_SB) {
    Device (SMTS) {
        Name (_HID, "INTC1081")
        Name (_STR, ...)

        Method (_DSM, ...) {
            ...
        }
    }
}
```

2.3.1 _DSM Method

The _DSM method for an SMTS device is described in this section.

**Input Parameters:**

Arg0 – UUID (set to 75191659-8178-4D9D-B88F-AC5E5E93E8BF)

Arg1 – Revision ID (set to 1)

Arg2 – Function Index

Table 2-7. SMTS Device Function Index

<table>
<thead>
<tr>
<th>Function Index</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Standard _DSM query function</td>
</tr>
<tr>
<td>1</td>
<td>Set MM Log Level (refer to Section 2.3.3)</td>
</tr>
<tr>
<td>2</td>
<td>Get MM Log Level (refer to Section 2.3.4)</td>
</tr>
<tr>
<td>3</td>
<td>Get MM Log Data Info (refer to Section 2.3.5)</td>
</tr>
<tr>
<td>All other value</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

Arg3 – A package containing parameters for the function specified by the UUID, Revision ID and Function Index. The layout of the package for each command along with the corresponding output is illustrated in the respective Function Index description sections.
2.3.2 Return Status Values

Same status code values are used as Section 2.2.2.

2.3.3 Function Index 1 – Set MM Log Level

This function allows software to set MM log level in MM Telemetry Service.

Log Level Definition:

- 0 – Error Message
- 1 – Warning Message
- 2 – Informational Message
- 4 – Verbose (Detailed message)

Function Input

Package {
    Log Level  // Integer
}

Table 2-8. Set MM Log Level – Input Package Values

<table>
<thead>
<tr>
<th>Field</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log Level</td>
<td>Integer</td>
<td>The MM telemetry log level to be set. Any log message with level equal to or less than Log Level (that is, of equal or higher priority) will be recorded by MM Telemetry Service. A log message with levels greater than Log Level will be discarded.</td>
</tr>
</tbody>
</table>

Function Output

A package as described next.

Package {
    Status       // Integer
    Extended Status // Integer
}

Table 2-9. Set MM Log Level – Return Package Values

<table>
<thead>
<tr>
<th>Field</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>Integer</td>
<td>Defined in Table 2-2</td>
</tr>
<tr>
<td>Extended Status</td>
<td>Integer</td>
<td>Implementation specific</td>
</tr>
<tr>
<td>Log Level</td>
<td>Integer</td>
<td>Configured log level value in MM Telemetry Service.</td>
</tr>
</tbody>
</table>
2.3.4 Function Index 2 – Get MM Log Level

This function allows software to get current MM log level from MM Telemetry Service.

Function Input
None

Function Output
A package as described next.

Package {
    Status     // Integer
    Extended Status   // Integer
    Log Level     // Integer
}

Table 2-10. Get MM Log Level – Return Package Values

<table>
<thead>
<tr>
<th>Field</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>Integer</td>
<td>Defined in Table 2-2</td>
</tr>
<tr>
<td>Extended Status</td>
<td>Integer</td>
<td>Implementation specific</td>
</tr>
<tr>
<td>Log Level</td>
<td>Integer</td>
<td>Current log level in MM Telemetry Service.</td>
</tr>
</tbody>
</table>

2.3.5 Function Index 3 – Get MM Log Data Info

This function allows software to get MM Log Data from MM Telemetry Service. The function can return MM Runtime Update execution log data or history log data, depends on the data type value of the function input.

- Execution Log: MM execution log data generated since previous call of Start MM Runtime Update process (Section 2.2.5). The log data format is implementation specific like the BIOS serial log messages.
- History Information: MM Runtime update history information. The log data format is implementation specific like the BIOS serial log messages, but these log messages are not reset on MM Runtime Update process.

OS shall follow the Appendix A (OS Agent Sample Algorithm) described algorithm to retrieve delta MM Log Data with this function.

MM Log Data is generated by MM Runtime update driver, SMI handler or other MM modules, and consumed when this function is called. The detailed definition of the log entry format is out of scope and covered by MM Runtime Update – Telemetry Specification.
Function Input

DSM method Revision 1 doesn’t support function input and always return Execution Log.

DSM method Revision 2 supports below function input:

```
Package {
    Data Type // Integer
}
```

Table 2-11. Get MM Log Data Info – Input Package Values

<table>
<thead>
<tr>
<th>Field</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Type</td>
<td>Integer</td>
<td>The MM telemetry log data type to get.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 – Execution Log.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 – History Information.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Other value – Reserved.</td>
</tr>
</tbody>
</table>

Function Output

A package as described next.

```
Package {
    Status // Integer
    Extended Status // Integer
    Maximum Data Chunk Size // Integer
    Data Chunk1 Address Low // Integer
    Data Chunk1 Address High // Integer
    Data Chunk1 Size // Integer
    Data Chunk2 Address Low // Integer
    Data Chunk2 Address High // Integer
    Data Chunk2 Size // Integer
    Rollover Count // Integer
    Telemetry Service Reset Count // Integer
}
```
### Table 2-12. Get MM Log Data – Return Package Values

<table>
<thead>
<tr>
<th>Field</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>Integer</td>
<td>Defined in <a href="#">Table 2-2</a></td>
</tr>
<tr>
<td>Extended Status</td>
<td>Integer</td>
<td>Implementation specific</td>
</tr>
<tr>
<td>Maximum Data Chunk Size</td>
<td>Integer</td>
<td>Maximum supported size of data of all Data Chunks combined</td>
</tr>
<tr>
<td>Data Chunk1 Address Low</td>
<td>Integer</td>
<td>Low 32-bit physical address of the telemetry data chunk1 starting address.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> This field is not applicable to OOB MM Update.</td>
</tr>
<tr>
<td>Data Chunk1 Address High</td>
<td>Integer</td>
<td>High 32-bit physical address of the telemetry data chunk1 starting address.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> This field is not applicable to OOB MM Update.</td>
</tr>
<tr>
<td>Data Chunk1 Size</td>
<td>Integer</td>
<td>Data size in bytes of the telemetry data chunk1 buffer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> This field is not applicable to OOB MM Update.</td>
</tr>
<tr>
<td>Data Chunk2 Address Low</td>
<td>Integer</td>
<td>Low 32-bit physical address of the telemetry data chunk2 starting address.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> This field is not applicable to OOB MM Update.</td>
</tr>
<tr>
<td>Data Chunk2 Address High</td>
<td>Integer</td>
<td>High 32-bit physical address of the telemetry data chunk2 starting address.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> This field is not applicable to OOB MM Update.</td>
</tr>
<tr>
<td>Data Chunk2 Size</td>
<td>Integer</td>
<td>Data size in bytes of the telemetry data chunk2 buffer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> This field is not applicable to OOB MM Update.</td>
</tr>
<tr>
<td>Rollover Count</td>
<td>Integer</td>
<td>Number of times MM telemetry data buffer is overwritten since telemetry buffer reset.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> MM telemetry service uses a buffer to store telemetry data that has maximum capacity limitation. If data being logged reaches the maximum capacity, old telemetry data is overwritten by new one. This is called overflow condition.</td>
</tr>
<tr>
<td>Telemetry Service Reset Count</td>
<td>Integer</td>
<td>Number of times telemetry services resets that results in Rollover Count and Data Chunk buffers are reset</td>
</tr>
</tbody>
</table>

**Note:** Data Chunk2 Address shall be equal to Data Chunk1 Address + Data Chunk1 Size.

Use of Data Chunk1 and Data Chunk2 allows implementation flexibility for use of circular buffer and reduce merging overhead for the BIOS.
A Appendix (OS Agent Sample Algorithm)

This appendix provides sample algorithms for OS agent to retrieve MM Log Data.

A.1 Telemetry Buffer Data Retrieval

The following pseudo code outlines the OS agent algorithm for retrieving currently existing data from MM telemetry service. If the telemetry buffer gets full, most recent log data will overwrite old log data.

- Call ACPI DSM Get MM Log Data Info function
- Copy Data Chunk1 Size data starting from Data Chunk1 Address to OS owned destination storage
- Append Data Chunk2 Size data starting from Data Chunk2 Address to OS owned destination storage
- If Rollover Count is non-zero, overflow condition is occurred
- Call Get MM Log Data Info function again
- If data changed between Get MM Log Data Info calls, SMI had occurred and new Log Data was added during the processing. Repeat previous steps until data match

A.2 Delta Telemetry Data Retrieval

The following pseudo code outlines the OS agent algorithm for retrieving new MM Log Data since previous data retrieval:

First time access after system boot or on Telemetry Service Reset:

1. Follow the steps outlines Section A.1
2. Save the Get MM Log Data Info return data as previous parameters

Subsequent access (either polling or invoked again):

1. Call ACPI DSM Get MM Log Data Info function to determine the delta from delta starting parameters
2. If no parameter change
   - no new data present
3. Else If Rollover Count == previous Rollover Count && Telemetry Service Reset Count == previous Telemetry Service Reset Count
   - Data Start Address = previous Data Chunk2 Address + previous Data Chunk2 Size
   - Data Size = Data Chunk2 Size – previous Data Chunk2 Size
   - Copy data from Data Start Address of Data Size to destination storage
   - No data loss, indicate as no overflow condition
4. Else If Rollover Count == previous Rollover Count + 1 && Telemetry Service Reset Count == previous Telemetry Service Reset Count
   • If Data Chunk2 Size <= previous Data Chunk2 Size (condition where new log data pointer has not moved past old log data pointer)
     o Data Start Address = previous Data Chunk2 Address + previous Data Chunk2 Size
     o Data Size = Maximum Data Chunk Size – Data Start Address
     o Copy data from Data Start Address of Data Size to destination storage
     o Data Start Address = Data Chunk2 Address
     o Data Size = Data Chunk2 Size
     o Append data from Data Start Address of Data Size to destination storage
     o No data loss, indicate as no overflow condition
   • If Data Chunk2 Size > previous Data Chunk2 Size (condition where new log data pointer has not moved past old log data pointer)
     o Copy Data Chunk1 Size data starting from Data Chunk1 Address to destination storage
     o Append Data Chunk2 Size data starting from Data Chunk2 Address to destination storage
     o Data loss, indicate as overflow condition
5. Else If Rollover Count >= delta starting Rollover Count + 1 && Telemetry Service Reset Count == delta starting Telemetry Service Reset Count
   • Copy Data Chunk1 Size data starting from Data Chunk1 Address to destination storage
   • Append Data Chunk2 Size data starting from Data Chunk2 Address to destination storage
   • Data loss, indicate as overflow condition
6. Else If Telemetry Service Reset Count != delta starting Telemetry Service Reset Count
   • Data may have lost between telemetry service reset
   • Go to “First time access after system boot or on Telemetry Service Reset” flow to start new delta log
7. Call ACPI DSM Get MM Log Data Info function again to check no data changed from previous call
8. If any Data Chunk Address, Data Chunk Size, Rollover Count or Telemetry Service Reset Count values changed between Get MM Log Data Info calls in step #1 and #7, SMI occurred and new Log Data is added during the processing. Go to step #1 to repeat the previous steps until data matches, or exit after several retries.
9. Save step #7 Get MM Log Data Info return data as new previous parameters