**Document Number: DSP0247**

**Date: 2009-04-23**

**Version: 1.0.0**

# Platform Level Data Model (PLDM) for BIOS Control and Configuration Specification

**Document Type: Specification**

**Document Status: DMTF Standard**

**Document Language: E**

Implementation of certain elements of this standard or proposed standard may be subject to third party patent rights, including provisional patent rights (herein "patent rights"). DMTF makes no representations to users of the standard as to the existence of such rights, and is not responsible to recognize, disclose, or identify any or all such third party patent right, owners or claimants, nor for any incomplete or inaccurate identification or disclosure of such rights, owners or claimants. DMTF shall have no liability to any party, in any manner or circumstance, under any legal theory whatsoever, for failure to recognize, disclose, or identify any such third party patent rights, or for such party’s reliance on the standard or incorporation thereof in its product, protocols or testing procedures. DMTF shall have no liability to any party implementing such standard, whether such implementation is foreseeable or not, nor to any patent owner or claimant, and shall have no liability or responsibility for costs or losses incurred if a standard is withdrawn or modified after publication, and shall be indemnified and held harmless by any party implementing the standard from any and all claims of infringement by a patent owner for such implementations.

For information about patents held by third-parties which have notified the DMTF that, in their opinion, such patent may relate to or impact implementations of DMTF standards, visit http://www.dmtf.org/about/policies/disclosures.php.

# CONTENTS

# Figures

# Tables

# Foreword

The *Platform Level Data Model (PLDM) for BIOS Control and Configuration Specification* (DSP0247) was prepared by the Platform Management Components Intercommunications (PMCI) Working Group.

DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems management and interoperability.

# Introduction

The BIOS management and boot control Common Information Model (CIM) profiles define the remote management aspects of the BIOS configuration and control. A management controller that is exposing the remote management aspects of the BIOS performs internal platform communications with the BIOS to exchange the data related to BIOS configuration and control. PLDM for BIOS Control/Configuration defines the data structures and messages for communicating BIOS settings, BIOS attributes, boot configurations, and boot order settings. The *Platform Level Data Model (PLDM) for BIOS Control and Configuration Specification* is complementary to the BIOS management and boot control profiles.

# Platform Level Data Model (PLDM) for BIOS Control and Configuration Specification

## 1 Scope

The scope of this specification is to define the data structures and commands for the internal platform communications between a management controller and the BIOS to exchange the data related to BIOS configuration and control. This specification defines the data structures and messages for communicating BIOS settings, BIOS attributes, boot configurations, and boot order settings.

This specification is complementary to the BIOS management and boot control profiles.

## 2 Normative References

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

### 2.1 Approved References

DMTF DSP0240, *Platform Level Data Model (PLDM) Base Specification,* http://www.dmtf.org/standards/published_documents/DSP0240_1.0.0.pdf

DMTF DSP0245, *Platform Level Data Model (PLDM) IDs and Codes,* http://www.dmtf.org/standards/published_documents/DSP0245_1.0.0.pdf

DMTF DSP1012, *Boot Control Profile 1.0.0,* http://www.dmtf.org/standards/published_documents/DSP1012_1.0.0.pdf

DMTF DSP1061, *BIOS Management Profile 1.0.0,* http://www.dmtf.org/standards/published_documents/DSP1061_1.0.0.pdf

### 2.2 Other References

ISO/IEC Directives, Part 2, *Rules for the structure and drafting of International Standards,* http://isotc.iso.org/livelink/livelink.exe?func=ll&objId=4230456&objAction=browse&sort=subtype

OMG, *Unified Modeling Language (UML) from the Open Management Group (OMG),* http://www.uml.org/

## 3 Terms and Definitions

Refer to DSP0240 for terms and definitions that are used across the PLDM specifications. For the purposes of this document, the following additional terms and definitions apply.

**3.1**

**BIOS**

**Basic Input Output System**

refers to a piece of firmware code that runs on a computer system during startup to enable the computer to start the operating system and to communicate with the various devices in the system, such as disk drives, keyboard, monitor, printer, and communications ports

**3.2**

**BIOS Attribute**

represents a specific BIOS parameter or configuration setting

Each BIOS attribute is represented by a name, type, and type-specific metadata and values.

**3.3**

**BIOS Table**

is defined in this specification as a data structure that carries a set of BIOS attribute specific information

**3.4**

**BIOS Attribute Table**

is a BIOS table that contains attribute name handles, attribute types, type-specific metadata, type-specific possible values (if any), and default values

**3.5**

**BIOS Attribute Pending Value Table**

is a BIOS table that contains all the pending values of the BIOS attributes and settings

Each entry in this table contains the attribute handle, the attribute type, and pending values.

**3.6**

**BIOS Attribute Value Table**

is a BIOS table that contains all the current values of the BIOS attributes and settings

Each entry in this table contains the attribute handle, the attribute type, and current values.

**3.7**

**BIOS String Table**

is a BIOS table that contains all the BIOS strings including attribute names, and pre-configured strings used in representing the values of the attributes

Each string in the BIOS String Table has an associated unique handle.

# 4 Symbols and Abbreviated Terms

Refer to DSP0240 for symbols and abbreviated terms that are used across the PLDM specifications. For the purposes of this document, the following additional symbols and abbreviated terms apply.

**4.1**

**BIOS**

Basic Input Output System

# 5 Conventions

Refer to DSP0240 for conventions, notations, and data types that are used across the PLDM specifications.

# 6 PLDM for BIOS Control and Configuration Overview

The BIOS management and boot control CIM profiles define the remote management aspects of the BIOS configuration and control. In the context of this specification, the term Management Controller (MC) refers to a management controller that performs internal communications with the BIOS to exchange the BIOS configuration and control related data. The PLDM for BIOS Control/Configuration defines the data structures and messages for communicating BIOS settings, BIOS attributes, boot configurations, and boot order settings. The PLDM for BIOS Control/Configuration is complementary to the BIOS management and boot control profiles.

## 6.1 BIOS Attribute Update Models

When the MC exposes BIOS attributes to the remote management console, two models for updating the BIOS attributes exist:

- Immediate update model: In this model, the MC acts as a pass-through device. When a remote management console updates a BIOS attribute on the MC, the MC immediately updates the BIOS attributes (typically done by providing the updates to the BIOS for processing and responding to the console only after the BIOS has processed the updates). In some implementations, the MC may directly update the BIOS settings (for example, writing to the CMOS directly). This specification defines the data structures and commands that support the immediate update model where the MC is providing updates to the BIOS using PLDM messages.

- Deferred update model: In this model, the BIOS attribute changes are not done immediately but are cached as pending changes that do not take effect until the next time the BIOS runs. For example, the MC can act as a cache of the BIOS settings and attributes. When a remote management console updates a BIOS attribute on the MC, the MC caches the attribute change initiated remotely as the pending value and responds to the remote management console that the change is pending. The next time the BIOS runs, the MC provides the BIOS attribute change to the BIOS. The BIOS processes the update and informs the MC whether it accepted or rejected the change. This specification defines the data structures and commands that support the deferred update model where the MC is caching the BIOS attribute changes and PLDM messages are used when the BIOS runs to transfer attribute metadata, values, and updates.

## 6.2 BIOS and MC Communication Model

In this model, the BIOS is the owner of the BIOS attributes that get used by the system. The MC maintains a cached copy of the attributes. The local attribute changes are communicated between the BIOS and the MC using either a push or a pull model. In the push model, the BIOS control and configuration data transfer is initiated by the sender without being explicitly requested by the receiving entity. In the pull model, the transfer of the BIOS control and configuration data is requested by a receiving entity. The BIOS initiating the transfer of the local attributes changes to the MC is an example of push model. The BIOS querying the MC for pending attribute changes made by the remote management console is an example of the pull model.

In a typical implementation, the MC communicates the changes made by the remote management console to the BIOS. The BIOS either accepts or rejects the changes made by the remote management entity and communicates the acceptance or rejection of the pending changes to the MC. If the BIOS accepts the attribute changes made by the remote entity, the MC makes the changes permanent to its copy of the BIOS attributes. If the BIOS rejects the attribute changes, the MC discards the pending changes.

The BIOS settings and configuration can also be modified locally. In this case, the BIOS propagates the locally made changes to the MC. The BIOS configuration and control data is generally communicated between the BIOS and the MC using PLDM messages.

Additionally, the same PLDM messages may also be used to transfer BIOS configuration and control data between two management controllers. This may be done in a configuration where one of the management controllers is interfacing with the BIOS and the other management controller is interfacing with the remote management console. The BIOS data of interest are BIOS attributes, BIOS passwords, BIOS settings, and so on.

Below is an example of flow of operations to change a BIOS attribute remotely using the deferred update model. In this example, the MC is acting as a MAP that implements the *BIOS Management Profile* (DSP1061) and exposes the BIOS attributes to the remote management console. The BIOS pushes the BIOS configuration data, and pulls the configuration data changes from the MC using the commands described in this specification.

1) Initially, the BIOS provides the MC with a list of attributes that it wants to expose to the remote management console in the out-of-band environment. The MC creates the instances of the classes and associations to represent these BIOS attributes.
2) A remote management console discovers what BIOS attributes are exposed by the management service running on the MC.
3) For each BIOS attribute that it wants to change, the remote management console executes the SetBIOSAttribute() method (an extrinsic method defined in the *BIOS Management Profile,* DSP1061) to change the BIOS attribute remotely.
4) For each BIOS attribute change, the MC processes the SetBIOSAttribute() method and caches the BIOS attribute change as the pending value.
5) The next time the BIOS runs, it queries the MC to inquire what BIOS attributes have changed. If one or more BIOS attributes have pending values, the MC provides the list of BIOS attributes with pending values.
6) If pending values exist, the BIOS gets the pending values of the BIOS attributes.
7) For each BIOS attribute that has a pending value, the BIOS accepts or rejects the change and informs the MC accordingly.
8) If the BIOS accepts the pending value of a BIOS attribute, the MC sets the current value of the BIOS attribute to the pending value. If the BIOS rejects the pending value, the MC discards the pending value for that particular BIOS attribute.

In the immediate update model, the MC will typically push the BIOS configuration data changes as soon as they were received from the remote management console. The MC returns the success or failure status to the remote management console after performing the changes locally using PLDM messages. In most implementations, the immediate updates will happen only when the system firmware or BIOS is executing.

## 6.3 PLDM Components for BIOS Control and Configuration

The *Platform Level Data Model (PLDM) for BIOS Control and Configuration Specification* encompasses the following data structures and operations:

- BIOS attribute related data structure definitions, including BIOS tables
- BIOS attribute data structure definition for boot configurations and boot source settings
- BIOS attribute data structure definition for BIOS configuration settings
- PLDM commands for the BIOS table and attribute data transfers

The ordering and dependency among attribute data transfers is not covered by the PLDM. The PLDM for BIOS Control and Configuration commands simply transfer the BIOS attribute changes or the entire BIOS table. The PLDM for BIOS Control and Configuration definition does not track or control the order in which the BIOS or MC applies the changes. Also, the aggregation of the BIOS attributes' data transfer is not handled at the PLDM level.

## 6.4 BIOS Attribute Types

The *BIOS Management Profile* defines four types of BIOS attributes: BIOSEnumeration, BIOSString, BIOSPassword, and BIOSInteger. In addition, the *Boot Control Profile* (DSP1012) defines the boot configuration and boot order representations. The *BIOS Management Profile* also defines a collection of attributes as a separate class. For the PLDM for BIOS Control and Configuration, the following attribute types are defined:

- BIOSEnumeration represents a BIOS attribute that can have a value from a set of possible values.
- BIOSString represents a string that the BIOS uses. Each BIOS string is characterized by the minimum length of the string, the maximum length of the string, and the type of the string.
- BIOSPassword represents a string that has an additional characteristic: the password encoding type.
- BIOSInteger represents a BIOS integer. Each BIOS integer has a lower bound and an upper bound on the values that it can take.
- BIOSBootConfigSetting represents a boot configuration setting that includes information about the boot sources. The number of boot sources for a given boot configuration is within a range specified for the boot configuration.
- BIOSCollection represents a collection of the BIOS attributes.
- BIOSConfigSet represents the types of BIOS configuration sets that the BIOS supports.

## 6.5 BIOS String, Attribute, and Value Tables

Typically, the BIOS maintains the BIOS settings and attributes in a table-like format. The BIOS attribute names and values are represented using strings. The BIOS strings seldom change. Furthermore, most of the BIOSEnumeration attributes share the same strings. Therefore, a smaller handle can be used to refer to a string in a PLDM command. This approach increases the efficiency of the BIOS attribute data transfer. The PLDM for BIOS Control and Configuration defines a handle-based model for the BIOS tables. The following tables are defined for the PLDM for BIOS Control and Configuration data transfer:

- BIOS String Table, which contains all the attribute names and all the preconfigured strings used in representing the values of the attributes
- BIOS Attribute Table, which contains the attribute name handles, attribute types, type-specific metadata, type-specific possible values (if any), and default values
- BIOS Attribute Value Table, which contains all the current values of the BIOS attributes
- BIOS Attribute Pending Value Table, which contains all the pending values of the BIOS attributes

## 6.6 BIOS Table Tags

The BIOS tables change infrequently. Therefore, the transfer of BIOS tables should be avoided when the MC has the latest copies of the BIOS tables. The PLDM for BIOS Control and Configuration defines a tag-based mechanism that can be used to identify whether two BIOS tables are identical or not. This tag mechanism is defined mainly to improve the efficiency of the deferred update model. For example, the BIOS can query the BIOS tables maintained by the MC to determine whether the tables need an update or not. If the update is needed, then the BIOS copies the BIOS tables and sets the BIOS table tags after updating the copies on the MC.

The BIOS uses a BIOS table tag to identify a particular copy of the BIOS table. The BIOS table tag can be a simple identifier (like version information) or an integrity checksum of the entire table. The PLDM for BIOS Control and Configuration does not dictate any specific value or algorithm for the BIOS table tags.

The MC can treat the BIOS table tags as opaque identifiers of the BIOS tables and store the BIOS table tags provided by the BIOS.

When the BIOS updates a BIOS table on the MC, the BIOS provides the new table tag to the MC after updating the table. During startup, the BIOS retrieves the table tags from the MC before the BIOS table transfer to determine which copies of the BIOS tables are not up-to-date on the MC. This allows the BIOS to transfer only the BIOS tables that are not up-to-date on the MC.

## 6.7 Authentication Model

This specification does not define any specific authentication model between the BIOS and MC for the control and configuration data transfer using PLDM messages. Any authentication model would need to be layered on top of the PLDM for BIOS control and configuration and is outside the scope of this specification.

## 6.8 Restoring BIOS Defaults

The BIOS can have multiple types of default sets (for example, factory and fail-safe default settings). The BIOS keeps track of all the default values. The remote management console can restore BIOS default configurations by using the *BIOS Management Profile*. The MC is not required to store all the BIOS default values. At the PLDM level, the BIOS provides the information about the types of defaults to the MC in an attribute with an array of possible string values that provide the description of the default sets. The MC exposes this information to the remote management console. When the remote management console specifies restoration to a specific type of the BIOS default set, the MC conveys that change to the BIOS as the pending value of the attribute. The BIOS then treats it as the pending attribute change and informs the MC whether the BIOS accepted or rejected the change to the default set.

If the MC maintains a local copy of the BIOS tables, it is the responsibility of the BIOS to update those tables after the BIOS defaults have been restored.

The BIOSConfigSet attribute defined in this specification is used to represent one or more BIOS configurations that can be used by the console to restore the BIOS defaults to a particular configuration.

# 7 BIOS Tables

This section describes the BIOS tables used in mapping the string values, attribute metadata, and attribute values. Because a typical BIOS implementation has hundreds of strings, a 16-bit handle should be sufficient for the PLDM representation of the strings.

## 7.1 BIOS String Table

The BIOS String Table contains all the attribute names and all the preconfigured strings used in representing the values of the attributes.

The general structure of the BIOS String Table is shown in Table 1.

**Table 1 – General Structure of the BIOS String Table**

| BIOS String Handle | BIOS String |
|---|---|
| Handle 1 | String 1 |
| Handle 2 | String 2 |
| … | … |

The BIOS String Table representation in PLDM is described in Table 2.

**Table 2 – PLDM Representation of BIOSStringTableData**

| Byte | Type | Field |
|---|---|---|
| 0:1 | uint16 | **BIOSStringHandle[0]**<br>A handle that is used to identify the first string in the BIOS String Table |
| 2:3 | uint16 | **BIOSStringLength[0]**<br>The length of the first string in bytes |
| Variable | | **BIOSString[0]**<br>The first string |
| | uint16 | **BIOSStringHandle[1]**<br>A handle that is used to identify the second string in the BIOS String Table |
| | uint16 | **BIOSStringLength[1]**<br>The length of the second string in bytes |
| Variable | | **BIOSString[1]**<br>The second string |
| … | … | **…** |
| Variable | … | **Pad**<br>0 to 3 number of pad bytes. The value stored in each pad byte is 0x00.<br>The transmitter can compute the number of pad bytes from the BIOSStringTableData by using the following algorithm:<br>Let L be the total number of bytes in the BIOSStringTableData excluding the pad and the integrity checksum.<br>if (L modulo 4 == 0) then NumPadBytes = 0; else NumPadBytes = 4 – L modulo 4;<br>The receiver can compute the number of pad bytes from the BIOSStringTableData by using the following algorithm. In the algorithm, the receiver parses BIOSStringTableData until the remaining bytes are less than 8. When it reaches that stage, the remaining bytes contain the pad bytes and four bytes of data integrity checksum.<br>Let L be the total number of bytes in the BIOSStringTableData including the pad and the integrity checksum.<br>RemBytes = L;<br>i = 0;<br>while (RemBytes >= 8)<br>{<br>    Process the ith string in the table;<br>    RemBytes = RemBytes - 4 - BIOS String i Length;<br>    i = i+1;<br>}<br>NumPadBytes = RemBytes modulo 4; |
| | uint32 | **BIOSStringTableIntegrityChecksum**<br>Integrity checksum on the BIOSStringTableData shown above including the pad bytes (if any)<br>For this specification, the CRC-32 algorithm with the polynomial $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1$ (same as the one used by IEEE 802.3) must be used for the integrity checksum computation. The CRC computation involves processing a byte at a time with the least significant bit first. |

The following rules apply to all the strings in the BIOS String Table:

1. All the strings in the BIOS String Table shall be of one type. Note: The following well known string types are supported by this specification: ASCII, Hex, UTF-8, UTF-16LE (UTF-16 Little Endian), or UTF-16BE (UTF-16 Big Endian).

2. The type of strings in the BIOS String Table can be explicitly set by using the PLDM command defined in Section 8.15. If the type of strings is not explicitly set after setting the BIOS String Table, then all strings provided in the BIOSStringTable shall be treated as ASCII strings.

3. All ASCII strings shall use a single-byte to represent each character.

4. All Hex strings shall use two-bytes to represent each character represented by two hex digits.

5. All UTF-8 strings shall use one to four bytes to represent each character.

6. All UTF-16 strings shall use 2 or 4 bytes to represent each character.

7. For all ASCII strings, ASCII code for a character shall be between 0x00 to 0x7F. The particular ASCII character encoding such as ISO646-US, ISO8859-1, and so on is not specified.

8. For Hex strings, each Hex digit is represented as a single character where each character is represented using an encoding of a numeral 0 to 9 or a letter A to F (lower-case or upper-case) using ASCII character format defined in ISO646-US. In the hex strings, the hex digits appear in the order from the most significant digit to the least significant digit starting at offset 0 in the string. For example, “1f” represents hex value 0x1f in hex string format.

The string handles used in the string table are unique per string. The assignment of string handle values is implementation specific and no specific handle values are reserved. For example, an implementation may use an array index into an array of strings as a string handle. An example BIOS String Table is shown in Table 3.

**Table 3 – Example BIOS String Table**

| BIOS String Handle | BIOS String |
|---|---|
| 0x0000 | "Enabled" |
| 0x0001 | "On" |
| 0x0002 | "Off" |
| .. | |
| .. | |
| 0x0020 | "NumLock LED" |
| 0x0021 | "USB Emulation" |
| … | … |

## 7.2 BIOS Attribute Table

The BIOS Attribute Table contains the attribute name handles, attribute types, type-specific values, and default values. The BIOS attribute information can be communicated by using the BIOS Attribute Table that contains the information about each BIOS attribute. Each BIOS attribute entry in the table contains the following information:

- Attribute Handle is a 16-bit handle that uniquely identifies a BIOS attribute. The assignment of attribute handle values is implementation specific and no specific handle values are reserved.
- Attribute Name Handle is a 16-bit string handle that uniquely identifies a BIOS string from the BIOS String Table that represents the name of the BIOS attribute. The assignment of string handle values is implementation specific and no specific handle values are reserved.
- Attribute Type represents the type of the BIOS attribute. See Section 6.4 for the definitions of the different types of BIOS attributes.
- Type-specific Metadata contains one or more fields that describe type-specific properties of the BIOS attribute.
- Type-specific Possible Values contain one more possible legal or accepted values of the BIOS attribute. These values are applicable only for the BIOS attribute types BIOSEnumeration, BIOSBootConfigSetting, and BIOSConfigSet.
- Type-specific Default Values are represented by one or more values. Type-specific default values are applicable only for BIOS attribute types BIOSEnumeration, BIOSString, BIOSPassword, and BIOSInteger. For a BIOS attribute of type BIOSEnumeration, type-specific default values are represented by one or more entries into the array of the possible value.

The general structure of the BIOS Attribute Table is shown in Table 4.

**Table 4 – General Structure of BIOS Attribute Table**

| Attribute Handle | Attribute Type | Attribute Name Handle | Type-Specific Possible Values | Type-Specific Default Values |
|---|---|---|---|---|
| Attrib 0 Handle | Attrib 0 Type | Attrib 0 Name Handle | …. | … |
| Attrib 1 Handle | Attrib 1 Type | Attrib 1 Name Handle | … | … |
| … | … | … | … | … |

458 The BIOS Attribute Table representation in the PLDM is described in Table 5.

459 **Table 5 – PLDM Representation of BIOSAttributeTableData**

| Byte | Type | Field |
|---|---|---|
| 0:1 | uint16 | **AttributeHandle[0]**<br>A handle that is used to identify the first attribute in the BIOS Attribute Table |
| 2 | enum8 | **AttributeType[0]**<br>The type of the first attribute in the BIOS Attribute Table<br>Possible values:<br>{<br>BIOSEnumeration = 0x0,<br>BIOSString=0x1,<br>BIOSPassword=0x2,<br>BIOSInteger=0x3,<br>BIOSBootConfigSetting=0x4,<br>BIOSCollection=0x5,<br>BIOSConfigSet=0x6,<br>BIOSEnumerationReadOnly=0x80,<br>BIOSStringReadOnly=0x81,<br>BIOSPasswordReadOnly=0x82,<br>BIOSIntegerReadOnly=0x83,<br>BIOSBootConfigSettingReadOnly=0x84,<br>BIOSCollectionReadOnly=0x85<br>BIOSConfigSetReadOnly=0x86<br>}<br>Note: If it is not stated explicitly that an attribute is read-only, the BIOS attribute is considered as read-writable. |
| 3:4 | uint16 | **AttributeNameHandle[0]**<br>A handle that is used to identify the name of the first attribute in the BIOS Attribute Table. This handle points to a string in the BIOS String Table. |
| Variable | | **AttributeType[0] specific fields (see Table 6 through Table 12) for the first attribute** |
| | uint16 | **AttributeHandle[1]**<br>A handle that is used to identify the second attribute in the BIOS Attribute Table |
| | enum8 | **AttributeType[1]**<br>The type of the second attribute in the BIOS Attribute Table |
| | uint16 | **AttributeNameHandle[1]**<br>A handle that is used to identify the name of the second attribute in the BIOS Attribute Table. This handle points to a string in the BIOS String Table. |
| | | **AttributeType[1] specific fields (see Table 6 through Table 12) for the second attribute** |
| … | … | **…** |

| Byte | Type | Field |
|---|---|---|
| Variable | … | **Pad**<br>0 to 3 number of pad bytes. The value stored in each pad byte is 0x00.<br>The transmitter can compute the number of pad bytes from the BIOSAttributeTableData by using the following algorithm:<br>Let L be the total number of bytes in the BIOSAttributeTableData excluding the pad and the integrity checksum.<br>if (L modulo 4 == 0) then NumPadBytes = 0; else NumPadBytes = 4 – L modulo 4;<br>The receiver can compute the number of pad bytes from the BIOSAttributeTableData by using the following algorithm. In the algorithm, the receiver parses the BIOSAttributeTableData until the remaining bytes are less than 8. When it reaches that stage, the remaining bytes contain the pad bytes and four bytes of data integrity checksum.<br>Let L be the total number of bytes in the BIOSAttributeTableData including the pad and the integrity checksum.<br>RemBytes = L;<br>i = 0;<br>while (RemBytes >= 8)<br>{<br>    Process the ith attribute in the table;<br>    RemBytes = RemBytes - 5 – Length of ith attribute type specific fields;<br>    i = i+1;<br>}<br>NumPadBytes = RemBytes modulo 4; |
| … | uint32 | **BIOSAttributeTableIntegrityChecksum**<br>Integrity checksum on the BIOSAttributeTableData shown above including the pad information.<br>For this specification, the CRC-32 algorithm with the polynomial $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1$ (same as the one used by IEEE 802.3) must be used for the integrity checksum computation. The CRC computation involves processing a byte at a time with the least significant bit first. |

The specific fields for the BIOSEnumeration and BIOSEnumerationReadOnly types are described in Table 6.

**Table 6 – Specific BIOS Attribute Table Fields for BIOSEnumeration and BIOSEnumerationReadOnly Types**

| Byte | Type | Field |
|---|---|---|
| 0 | uint8 | **NumberOfPossibleValues (N)**<br>Total number of possible values for this enumeration |
| 1:2 | uint16 | **PossibleValueStringHandle[0]**<br>A handle that is used to identify the first possible string for this enumeration. This handle points to a string in the BIOS String Table. |
| … | … | **…** |
|  | uint16 | **PossibleValueStringHandle[N-1]**<br>A handle that is used to identify the Nth possible string for this enumeration. This handle points to a string in the BIOS String Table. |

| Byte | Type | Field |
|---|---|---|
| … | uint8 | **NumberOfDefaultValues (M)**<br>Total number of default values for this enumeration. A value of 0 indicates that this enumeration has no default values. |
| | uint8 | **DefaultValueStringHandleIndex[0]**<br>An index into the array of the possible values of string handles for the first default value. This index points to an entry in the array of string handles representing possible values for this attribute provided in the BIOS Attribute Table. |
| … | … | **…** |
| | uint8 | **DefaultValueStringHandleIndex[M-1]**<br>An index into the array of the possible values of string handles for the Mth default value. This index points to an entry in the array of string handles representing possible values for this attribute provided in the BIOS Attribute Table. |

464 The specific fields for the BIOSString and BIOSStringReadOnly types are described in Table 7.

465 **Table 7 – Specific BIOS Attribute Table Fields for BIOSString and BIOSStringReadOnly Types**

| Byte | Type | Field |
|---|---|---|
| 0 | enum8 | **StringType**<br>The type of the string. It identifies the character encoding used for this string.<br>Possible values:<br>{Unknown=0x00, ASCII=0x01, Hex=0x02, UTF-8=0x03, UTF-16LE=0x04, UTF-16BE=0x05, Vendor Specific=0xFF} |
| 1:2 | uint16 | **MinimumStringLength**<br>The minimum length of the string in bytes. |
| 3:4 | uint16 | **MaximumStringLength**<br>The maximum length of the string in bytes. The value of MaximumStringLength shall be greater than or equal to the value of MinimumStringLength. |
| 5:6 | uint16 | **DefaultStringLength**<br>The length of the default string in bytes. A value of 0 indicates that this attribute has no default string. |
| Variable | | **DefaultString**<br>The default string itself |

466 The specific fields for the BIOSPassword and BIOSPasswordReadOnly types are described in Table 8.

467 **Table 8 – Specific BIOS Attribute Table Fields for BIOSPassword and BIOSPasswordReadOnly**
468 **Types**

| Byte | Type | Field |
|---|---|---|
| 0 | enum8 | **PasswordEncodingType**<br>The encoding that is used for this password<br>Possible values:<br>{ASCII=0x0, kbd=0x1, pin=0x2, UTF-8=0x03, UTF-16LE=0x04, UTF-16BE=0x05, Vendor Specific=0xFF}<br>See the CIM_BIOSPassword MOF for the description of the password encoding types. |
| 1:2 | uint16 | **MinimumPasswordLength**<br>The minimum length of the password in bytes |

| Byte | Type | Field |
|---|---|---|
| 3:4 | uint16 | **MaximumPasswordLength**<br>The maximum length of the password in bytes. The value of MaximumPasswordLength shall be greater than or equal to the value of MinimumPasswordLength. |
| 5:6 | uint16 | **DefaultPasswordLength**<br>The length of the default password in bytes. A value of 0 indicates that the default password is not available. |
| Variable | | **DefaultPassword**<br>The default password itself |

469 The specific fields for the BIOSInteger and BIOSIntegerReadOnly types are described in Table 9.

470 **Table 9 – Specific BIOS Attribute Table Fields for BIOSInteger and BIOSIntegerReadOnly Types**

| Byte | Type | Field |
|---|---|---|
| 0:7 | uint64 | **LowerBound**<br>The lower bound on the integer value |
| 8:15 | uint64 | **UpperBound**<br>The upper bound on the integer value. The value of UpperBound shall be greater than or equal to the value of LowerBound. |
| 16:19 | uint32 | **ScalarIncrement**<br>The scalar value that is used for the increments to this integer |
| 20:27 | uint64 | **DefaultValue**<br>The default value of the integer |

471 The specific fields for the BIOSBootConfigSetting and BIOSBootConfigSettingReadOnly types are
472 described in Table 10.

473 **Table 10 – Specific BIOS Attribute Table Fields for BIOSBootConfigSetting and**
474 **BIOSBootConfigSettingReadOnly Types**

| Byte | Type | Field |
|---|---|---|
| 0 | enum8 | **BootConfigType**<br>The type of the boot configuration<br>Possible values:<br>{<br>Unknown (0x00) – A template boot configuration whose type is not known. This configuration is not known to be default, next, or one-time.<br>Default (0x01) – The default configuration for the BIOS boot configuration.<br>Next (0x2) – The next boot configuration that is maintained across the boots.<br>DefaultAndNext (0x3) – The default configuration that is used as the next boot configuration that is maintained across the boots. In this case, the BIOS uses the same boot configuration for both default and next boot configuration.<br>Onetime (0x4) – The one-time boot configuration that is used for the next boot only.<br>DefaultAndOnetime (0x5) – The default configuration that is used as one-time boot configuration for the next boot only. In this case, the BIOS uses the same boot configuration for both default and one-time boot configuration.<br>} |

| Byte | Type | Field |
|---|---|---|
| 1 | enum8 | **SupportedOrderedAndFailThroughModes**<br>The ordered and fail-through modes supported for the boot configuration<br>Possible values:<br>{<br>UnorderedAndLimitedFailThrough (0x00) – Supports unordered boot sources with limited fail through,<br>UnorderedAndFailThrough (0x01) – Supports unordered boot sources with unlimited fail through,<br>OrderedAndLimitedFailThrough (0x02) – Supports ordered boot sources with limited fail through,<br>OrderedAndFailThrough (0x03) – Supports ordered boot sources with unlimited fail through<br>Unordered (0x04) – Supports unordered boot sources with unlimited or limited fail through. Note: This means that the OrderAndFailThroughMode can be set to either UnorderedAndLimitedfailThrough or UnorderedAndFailThrough,<br>Ordered (0x05) – Supports unordered and ordered boot sources with limited fail through. Note: This means that the OrderAndFailThroughMode can be set to either OrderedAndLimitedfailThrough or OrderedAndFailThrough,<br>LimitedFailThrough (0x06) – Supports unordered and ordered boot sources with limited fail through. Note: This means that the OrderAndFailThroughMode can be set to either UnorderedAndLimitedFailThrough or OrderedAndLimitedFailThrough,<br>FailThrough (0x07) – Supports unordered and ordered boot sources with unlimited fail through. Note: This means that the OrderAndFailThroughMode can be set to either UnorderedAndFailThrough or OrderedAndFailThrough,<br>All (0x08) – Supports all combinations. Note: This means that the OrderAndFailThroughMode can be set to any one of the following modes: UnorderedAndLimitedfailThrough, UnorderedAndFailThrough, OrderedAndLimitedfailThrough, and OrderedAndFailThrough.<br>} |
| 2 | uint8 | **MinimumNumberOfBootSourceSettings**<br>Specifies the minimum number of boot source settings that must be present in this boot configuration |
| 3 | uint8 | **MaximumNumberOfBootSourceSettings**<br>Specifies the maximum number of boot source settings that can be present in this boot configuration. The value of MaximumNumberOfBootSourceSettings shall be greater than or equal to the value of MinimumNumberOfBootSourceSettings. |
| 4 | uint8 | **NumberOfPossibleBootSourceSettings (N)**<br>Specifies the number of boot source settings that are possible for this boot configuration |
| 5:6 | uint16 | **PossibleBootSourceStringHandle[0]**<br>A handle to the first possible boot source setting string. This handle points to a string in the BIOS String Table. |
| … | … | **…** |
| 2N+3:2N+4 | uint16 | **PossibleBootSourceString Handle[N-1]**<br>A handle to the N$^{th}$ possible boot source setting string. This handle points to a string in the BIOS String Table. |

The BIOS Attribute Table can contain multiple boot configurations. When multiple BIOS attributes of type BIOSBootConfigSetting or BIOSBootConfigSettingReadOnly are provided in the BIOS Attribute Table, the following rules apply:

- At most one BIOS attribute shall exist among the BIOS attributes of type BIOSBootConfigSetting or BIOSBootConfigSettingReadOnly with BootConfigType set to Default, DefaultAndNext, or DefaultAndOnetime. In other words, the BIOS Attribute Table shall contain at most one default boot configuration.
- At most one BIOS attribute shall exist among the BIOS attributes of type BIOSBootConfigSetting or BIOSBootConfigSettingReadOnly with BootConfigType set to Next or DefaultAndNext. In other words, the BIOS Attribute Table shall contain at most one next boot configuration.
- At most one BIOS attribute shall exist among the BIOS attributes of type BIOSBootConfigSetting or BIOSBootConfigSettingReadOnly with BootConfigType set to Onetime or DefaultAndOnetime. In other words, the BIOS Attribute Table shall contain at most one one-time boot configuration.
- For the next boot, the boot configuration with BootConfigType set to Onetime or DefaultAndOnetime takes precedence over the boot configuration with BootConfigType set to Next or DefaultAndNext.

The specific fields for the BIOSCollection and BIOSCollectionReadOnly types are described in Table 11.

**Table 11 – Specific BIOS Attribute Table Fields for BIOSCollection and BIOSCollectionReadOnly Types**

| Byte | Type | Field |
|---|---|---|
| 0:1 | uint16 | **CollectionNameStringHandle**<br>A handle to the BIOS collection name string |
| 2 | uint8 | **MaximumNumberOfAttributes (N)**<br>The maximum number of BIOS attributes that belong to this collection |
| 3 | enum8 | **CollectionType**<br>The type of the BIOS collection<br>Possible values:<br>{<br>UnorderedAndUnmodifiable (0x00) – The attributes in this collection are not ordered. The collection cannot be modified.<br>UnorderedAndModifiable (0x01) – The attributes in this collection are not ordered. The collection can be modified.<br>OrderedAndUnmodifiable (0x02) – The attributes in this collection are ordered in ascending order. The collection cannot be modified.<br>OrderedAndModifiable (0x03) – The attributes in this collection are ordered in ascending order. The collection can be modified.<br>}<br>Note: A read-only BIOS collection means that all the attributes of the collection are read-only. |

The specific fields for the BIOSConfigSet and BIOSConfigSetReadOnly types are as described in Table 12.

NOTE: The BIOS needs to provide BIOSConfigSet to the MC if it wants to allow restoration of the BIOS defaults. Each configuration in the BIOS configuration set corresponds to a BIOS element. The BIOS Attribute Table shall contain at most one attribute of type BIOSConfigSet.

**Table 12 – Specific BIOS Attribute Table Fields for BIOSConfigSet and BIOSConfigSetReadOnly Types**

| Byte | Type | Field |
|---|---|---|
| 0 | uint8 | **NumberOfPossibleBIOSConfigurations (N)**<br>The number of possible BIOS configurations |
| 1:2 | uint16 | **PossibleBIOSConfigStringHandle[0]**<br>A handle to the first possible BIOS configuration string. This handle points to a string in the BIOS String Table. |
| … | … | … |
| 2N-1:2N | uint16 | **PossibleBIOSConfigStringHandle[N-1]**<br>A handle to the Nth possible BIOS configuration string. This handle points to a string in the BIOS String Table. |

## 7.3 BIOS Attribute Value Table

The BIOS Attribute Value Table contains all the current values of the BIOS attributes. Each BIOS attribute entry in this table contains

- Attribute Handle
- Attribute Type
- Current values

The general structure of the BIOS Attribute Value Table is shown in Table 13.

**Table 13 – General Structure of BIOS Attribute Value Table**

| Attribute Handle | Attribute Type | Type Specific Current Values |
|---|---|---|
| Attrib 0 Handle | Attrib 0 Type | … |
| Attrib 1 Handle | Attrib 1 Type | … |
| … | … | … |

The BIOS Attribute Value Table representation in PLDM is described in Table 14.

**Table 14 – PLDM Representation of BIOSAttributeValueTableData**

| Byte | Type | Field |
|---|---|---|
| 0:1 | uint16 | **AttributeHandle[0]**<br>A handle that is used to identify the first attribute in the BIOS Attribute Value Table. This handle points to an attribute in the BIOS Attribute Table. |
| 2 | enum8 | **AttributeType[0]**<br>The type of the first attribute in the BIOS Attribute Value Table<br>Possible values:<br>{<br>BIOSEnumeration = 0x0,<br>BIOSString=0x1,<br>BIOSPassword=0x2,<br>BIOSInteger=0x3, |

| Byte | Type | Field |
|---|---|---|
| | | BIOSBootConfigSetting=0x4,<br>BIOSCollection=0x5,<br>BIOSConfigSet=0x6,<br>BIOSEnumerationReadOnly=0x80,<br>BIOSStringReadOnly=0x81,<br>BIOSPasswordReadOnly=0x82,<br>BIOSIntegerReadOnly=0x83,<br>BIOSBootConfigSettingReadOnly=0x84,<br>BIOSCollectionReadOnly=0x85<br>BIOSConfigSetReadOnly=0x86<br>} |
| Variable | | **AttributeType[0] specific fields (see Table 15 through Table 21) for the first attribute** |
| | uint16 | **AttributeHandle[1]**<br>A handle that is used to identify the second attribute in the BIOS Attribute Value Table. This handle points to an attribute in the BIOS Attribute Table. |
| | enum8 | **AttributeType[1]**<br>The type of the second attribute in the BIOS Attribute Value Table |
| Variable | | **AttributeType[1] specific fields (see Table 15 through Table 21) for the second attribute** |
| | | |
| … | … | **…** |
| Variable | … | **Pad**<br>0 to 3 number of pad bytes. The value stored in each pad byte is 0x00.<br>The transmitter can compute the number of pad bytes from the BIOSAttributeValueTableData by using the following algorithm:<br>Let L be the total number of bytes in the BIOSAttributeValueTableData excluding the pad and the integrity checksum.<br>if (L modulo 4 == 0) then NumPadBytes = 0; else NumPadBytes = 4 – L modulo 4;<br>The receiver can compute the number of pad bytes from the BIOSAttributeValueTableData by using the following algorithm. In the algorithm, the receiver parses BIOS Attribute Value Table data until the remaining bytes are less than 8. When it reaches that stage, the remaining bytes contain the pad bytes and four bytes of data integrity checksum.<br>Let L be the total number of bytes in the BIOSAttributeValueTableData including the pad and the integrity checksum.<br>RemBytes = L;<br>i = 0;<br>while (RemBytes >= 8)<br>{<br>    Process the ith attribute in the table;<br>    RemBytes = RemBytes - 3 – Length of ith attribute type specific fields;<br>    i = i+1;<br>}<br>NumPadBytes = RemBytes modulo 4; |
| … | uint32 | **BIOSAttributeValueTableIntegrityChecksum**<br>Integrity checksum on the BIOSAttributeValueTableData shown above including the pad |

| Byte | Type | Field |
|---|---|---|
| | | information.<br>For this specification, the CRC-32 algorithm with the polynomial $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1$ (same as the one used by IEEE 802.3) must be used for the integrity checksum computation. The CRC computation involves processing a byte at a time with the least significant bit first. |

NOTE: The preceding representation can be used for transferring the entire BIOS Attribute Value Table or a subset of BIOS attribute values present in the BIOS Attribute Value Table.

The specific fields for the BIOSEnumeration and BIOSEnumerationReadOnly types are as described in Table 15.

**Table 15 – Specific BIOS Attribute Value Table Fields for BIOSEnumeration and BIOSEnumerationReadOnly Types**

| Byte | Type | Field |
|---|---|---|
| 0 | uint8 | **NumberOfCurrentValues (N)**<br>Total number of current values for this enumeration |
| 1 | uint8 | **CurrentValueStringHandleIndex[0]**<br>An index into the array of the possible values of string handles for the first current value. This index points to an entry in the array of string handles representing possible values for this attribute provided in the BIOS Attribute Table. |
| … | … | **…** |
| N | uint8 | **CurrentValueStringHandleIndex[N-1]**<br>An index into the array of the possible values of string handles for the Nth current value. This index points to an entry in the array of string handles representing possible values for this attribute provided in the BIOS Attribute Table. |

The string handle index in Table 15 is an 8-bit index into the array (provided in the BIOS Attribute Table) of the possible values of string handles for this attribute.

The specific fields for the BIOSString and BIOSStringReadOnly types are described in Table 16.

**Table 16 – Specific BIOS Attribute Value Table Fields for BIOSString and BIOSStringReadOnly Types**

| Byte | Type | Field |
|---|---|---|
| 0:1 | uint16 | **CurrentStringLength**<br>The length of the current string in bytes. A value of 0 indicates that the current string value is not set. |
| Variable | | **CurrentString**<br>The current string itself |

The specific fields for the BIOSPassword and BIOSPasswordReadOnly types are described in Table 17.

**Table 17 – Specific BIOS Attribute Value Table Fields for BIOSPassword and BIOSPasswordReadOnly Types**

| Byte | Type | Field |
|---|---|---|
| 0:1 | uint16 | **CurrentPasswordLength**<br>The length of the current password in bytes. A value of 0 indicates that the current password is set but not provided. |
| Variable | | **CurrentPassword**<br>The current password |

If the current password of a BIOS attribute of type BIOSPassword or BIOSPasswordReadOnly is not set, then the BIOS Attribute Value Table shall not contain that attribute in the BIOS Attribute Value Table.

The specific fields for the BIOSInteger and BIOSIntegerReadOnly types are described in Table 18.

**Table 18 – Specific BIOS Attribute Value Table Fields for BIOSInteger and BIOSIntegerReadOnly Types**

| Byte | Type | Field |
|---|---|---|
| 0:7 | uint64 | **CurrentValue**<br>The current value of the integer |

The specific fields for the BIOSBootConfigSetting and BIOSBootConfigSettingReadOnly types are described in Table 19.

**Table 19 – Specific BIOS Attribute Value Table Fields for BIOSBootConfigSetting and BIOSBootConfigSettingReadOnly Types**

| Byte | Type | Field |
|---|---|---|
| 0 | enum8 | **BootConfigType**<br>The type of the boot configuration<br>Possible values:<br>{<br>Unknown (0x00) – A template boot configuration whose type is not known. This configuration is not known to be default, next, or one-time.<br>Default (0x01) – The default configuration for the BIOS boot configuration.<br>Next (0x2) – The next boot configuration that is maintained across the boots.<br>DefaultAndNext (0x3) – The default configuration that is used as the next boot configuration that is maintained across the boots. In this case, the BIOS uses the same boot configuration for both default and next boot configuration.<br>Onetime (0x4) – The one-time boot configuration that is used for the next boot only.<br>DefaultAndOnetime (0x5) – The default configuration that is used as one-time boot configuration for the next boot only. In this case, the BIOS uses the same boot configuration for both default and one-time boot configuration.<br>} |

| Byte | Type | Field |
|---|---|---|
| 1 | enum8 | **OrderAndFailThroughMode**<br>Possible values:<br>{<br>UnorderedAndLimitedFailThrough (0x00) – The boot sources specified in the array below can be applied in any order. In the case of failure to boot from any boot sources specified in the array, other boot sources that are specified as possible boot sources for this attribute in the BIOS Attribute Table shall not be tried.<br>UnorderedAndFailThrough (0x01) – The boot sources specified in the array below can be applied in any order and in the case of failure to boot from the boot sources specified in the array, other boot sources that are specified as possible boot sources for this attribute in the BIOS Attribute Table can be tried in any order.<br>OrderedAndLimitedFailThrough (0x02) – The boot sources specified in the array below must be applied in the order specified in the array and in the case of failure to boot from the boot sources specified in the array, other boot sources that are specified as possible boot sources for this attribute in the BIOS Attribute Table shall not be tried.<br>OrderedAndFailThrough (0x03) – The boot sources specified in the array below must be applied in the order specified in the array and in the case of failure to boot from the boot sources specified in the array, other boot sources that are specified as possible boot sources for this attribute in the BIOS Attribute Table can be tried in any order.<br>} |
| 2 | uint8 | **NumberOfBootSourceSettings (N)**<br>Specifies the number of boot source settings that are in the current boot configuration |
| 3 | uint8 | **BootSourceStringHandleIndex[0]**<br>An index into the array of the possible values of string handles for the first boot source setting value. This index points to an entry in the array of string handles representing possible values for this attribute provided in the BIOS Attribute Table. |
| 4 | uint8 | **BootSourceStringHandleIndex[1]**<br>An index into the array of the possible values of string handles for the second boot source setting value. This index points to an entry in the array of string handles representing possible values for this attribute provided in the BIOS Attribute Table. |
| … | … | **…** |
| N+2 | uint8 | **BootSourceStringHandleIndex[N-1]**<br>An index into the array of the possible values of string handles for the Nth boot source setting value. This index points to an entry in the array of string handles representing possible values for this attribute provided in the BIOS Attribute Table. |

The string handle index in Table 19 is an 8-bit index into the array (provided in the BIOS Attribute Table) of the possible boot source string handles for this attribute.

The BIOS Attribute Value Table can contain settings for multiple boot configurations. When multiple BIOS attributes of type BIOSBootConfigSetting or BIOSBootConfigSettingReadOnly are provided in the BIOS Attribute Value Table, the following rules apply:

- At most one BIOS attribute shall exist among the BIOS attributes of type BIOSBootConfigSetting or BIOSBootConfigSettingReadOnly with BootConfigType set to Default, DefaultAndNext, or DefaultAndOnetime. In other words, the BIOS Attribute Value Table shall contain at most one default boot configuration.

- At most one BIOS attribute shall exist among the BIOS attributes of type BIOSBootConfigSetting or BIOSBootConfigSettingReadOnly with BootConfigType set to Next or DefaultAndNext. In other words, the BIOS Attribute Value Table shall contain at most one next boot configuration.
- At most one BIOS attribute shall exist among the BIOS attributes of type BIOSBootConfigSetting or BIOSBootConfigSettingReadOnly with BootConfigType set to Onetime or DefaultAndOnetime. In other words, the BIOS Attribute Value Table shall contain at most one onetime boot configuration.
- For the next boot, the boot configuration with BootConfigType set to Onetime or DefaultAndOnetime takes precedence over the boot configuration with BootConfigType set to Next or DefaultAndNext.

The specific fields for the BIOSCollection and BIOSCollectionReadOnly types are described in Table 20.

**Table 20 – Specific BIOS Attribute Value Table Fields for BIOSCollection and BIOSCollectionReadOnly Types**

| Byte | Type | Field |
|---|---|---|
| 0 | uint8 | **NumberOfAttributes (N)**<br>The number of BIOS attributes that belong to this BIOS collection |
| 1:2 | uintt16 | **AttributeHandle[0]**<br>A handle that is used to identify the first attribute in the current BIOS collection. This handle points to an attribute in the BIOS Attribute Table. |
| .. | uint16 | **….** |
| 2N-1:2N | uint16 | **AttributeHandle[N-1]**<br>A handle that is used to identify the N$^{th}$ attribute in the current BIOS collection. This handle points to an attribute in the BIOS Attribute Table. |

The specific fields for the BIOSConfigSet and BIOSConfigSetReadOnly types are described in Table 21. The BIOS Attribute Value Table shall contain at most one attribute of type BIOSConfigSet.

**Table 21 – Specific BIOS Attribute Value Table Fields for BIOSConfigSet and BIOSConfigSetReadOnly Types**

| Byte | Type | Field |
|---|---|---|
| 0 | uint8 | **CurrentConfigSetStringHandleIndex**<br>An index into the array of the possible values of BIOS configuration sets. This index points to an entry in the array of string handles representing possible values for this attribute provided in the BIOS Attribute Table. |

## 7.4 BIOS Attribute Pending Value Table

The BIOS Attribute Pending Value Table contains all the pending values of the BIOS attributes. Each BIOS attribute entry in this table contains

- Attribute Handle
- Attribute Type
- Pending values

The general structure of the BIOS Attribute Pending Value Table is shown in Table 22.

**Table 22 – General Structure of BIOS Attribute Pending Value Table**

| Attribute Handle | Attribute Type | Type Specific Pending Values |
|---|---|---|
| Attrib 0 Handle | Attrib 0 Type | … |
| Attrib 1 Handle | Attrib 1 Type | … |
| … | **…** | **…** |

The PLDM does not require the entity that maintains the BIOS Attribute Pending Value Table to perform attribute-level checking before accepting the pending values into the table.

The BIOS Attribute Pending Value Table representation in PLDM is described in Table 23.

**Table 23 – PLDM Representation of BIOSAttributePendingValueTableData**

| Byte | Type | Field |
|---|---|---|
| 0:1 | uint16 | **AttributeHandle[0]**<br>A handle that is used to identify the first attribute in the BIOS Attribute Pending Value Table. This handle points to an attribute in the BIOS Attribute Table. |
| 2 | enum8 | **AttributeType[0]**<br>The type of the first attribute in the BIOS Attribute Pending Value Table<br>Possible values:<br>{<br>BIOSEnumeration = 0x0,<br>BIOSString=0x1,<br>BIOSPassword=0x2,<br>BIOSInteger=0x3,<br>BIOSBootConfigSetting=0x4,<br>BIOSCollection=0x5,<br>BIOSConfigSet=0x6,<br>BIOSCollectionReadOnly=0x85<br>}<br>NOTE: By definition, the BIOS Attribute Pending Value Table contains BIOS attributes that are modified and pending approval from the BIOS. Thus, the BIOS Attribute Pending Value Table shall not contain read-only BIOS attributes except for the modifiable BIOS collection of read-only BIOS attributes. |
| Variable | | **AttributeType[0] specific fields (see Table 24 through Table 30)** |
| | uint16 | **AttributeHandle[1]**<br>A handle that is used to identify the second attribute in the BIOS Attribute Pending Value Table. This handle points to an attribute in the BIOS Attribute Table. |
| | enum8 | **AttributeType[1]**<br>The type of the second attribute in the BIOS Attribute Pending Value Table |
| Variable | | **AttributeType[1] specific fields (see Table 24 through Table 30)** |
| … | … | **…** |

| Byte | Type | Field |
|---|---|---|
| Variable | … | **Pad**<br>0 to 3 number of pad bytes. The value stored in each pad byte is 0x00.<br>The transmitter can compute the number of pad bytes from the BIOSAttributePendingValueTableData by using the following algorithm:<br>Let L be the total number of bytes in the BIOSAttributePendingValueTableData excluding the pad and the integrity checksum.<br>if (L modulo 4 == 0) then NumPadBytes = 0; else NumPadBytes = 4 – L modulo 4;<br>The receiver can compute the number of pad bytes from the BIOSAttributePendingValueTableData by using the following algorithm. In the algorithm, the receiver parses BIOS Attribute Pending Value Table data until the remaining bytes are less than 8. When it reaches that stage, the remaining bytes contain the pad bytes and four bytes of data integrity checksum.<br>Let L be the total number of bytes in the BIOSAttributePendingValueTableData including the pad and the integrity checksum.<br>RemBytes = L;<br>i = 0;<br>while (RemBytes >= 8)<br>{<br>    Process the ith attribute in the table;<br>    RemBytes = RemBytes - 3 – Length of ith attribute type specific fields;<br>    i = i+1;<br>}<br>NumPadBytes = RemBytes modulo 4; |
| … | uint32 | **BIOSAttributePendingValueTableIntegrityChecksum**<br>Integrity checksum on the BIOSAttributePendingValueTableData shown above including the pad information.<br>For this specification, the CRC-32 algorithm with the polynomial $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1$ (same as the one used by IEEE 802.3) must be used for the integrity checksum computation. The CRC computation involves processing a byte at a time with the least significant bit first. |

The specific fields for the BIOSEnumeration type are described in Table 24.

**Table 24 – Specific BIOS Attribute Pending Value Table Fields for the BIOSEnumeration Type**

| Byte | Type | Field |
|---|---|---|
| 0 | uint8 | **NumberOfPendingValues (N)**<br>Total number of pending values for this enumeration |
| 1 | uint8 | **PendingValueStringHandleIndex[0]**<br>An index into the array of the possible values of string handles for the first pending value. This index points to an entry in the array of string handles representing possible values for this attribute provided in the BIOS Attribute Table. |
| … | … | **…** |

| Byte | Type | Field |
|---|---|---|
| N | uint8 | **PendingValueStringHandleIndex[N-1]**<br>An index into the array of the possible values of string handles for the N$^{th}$ pending value. This index points to an entry in the array of string handles representing possible values for this attribute provided in the BIOS Attribute Table. |

577 The string handle index in Table 24 is an 8-bit index into the array (provided in the BIOS Attribute Table) of the possible values of string handles for this attribute.

579 The specific fields for the BIOSString type are described in Table 25.

580 **Table 25 – Specific BIOS Attribute Pending Value Table Fields for the BIOSString Type**

| Byte | Type | Field |
|---|---|---|
| 0:1 | uint16 | **PendingStringLength**<br>The length of the pending string in bytes |
| Variable | | **PendingString**<br>The pending string itself |

581 The specific fields for the BIOSPassword type are described in Table 26.

582 **Table 26 – Specific BIOS Attribute Pending Value Table Fields for the BIOSPassword Type**

| Byte | Type | Field |
|---|---|---|
| 0:1 | uint16 | **PendingPasswordLength**<br>The length of the pending password in bytes |
| Variable | | **PendingPassword**<br>The pending password |

583 The specific fields for the BIOSInteger type are described in Table 27.

584 **Table 27 – Specific BIOS Attribute Pending Value Table Fields for the BIOSInteger Type**

| Byte | Type | Field |
|---|---|---|
| 0:7 | uint64 | **PendingValue**<br>The pending value of the integer |

The specific fields for the BIOSBootConfigSetting type are described in Table 28.

**Table 28 – Specific BIOS Attribute Pending Value Table Fields for the BIOSBootConfigSetting Type**

| Byte | Type | Field |
|---|---|---|
| 0 | enum8 | **BootConfigType**<br>The type of the boot configuration<br>Possible values:<br>{<br>Unknown (0x00) – A template boot configuration whose type is not known. This configuration is not known to be default, next, or one-time.<br>Default (0x01) – The default configuration for the BIOS boot configuration.<br>Next (0x2) – The next boot configuration that is maintained across the boots.<br>DefaultAndNext (0x3) – The default configuration that is used as the next boot configuration that is maintained across the boots. In this case, the BIOS uses the same boot configuration for both default and next boot configuration.<br>Onetime (0x4) – The one-time boot configuration that is used for the next boot only.<br>DefaultAndOnetime (0x5) – The default configuration that is used as one-time boot configuration for the next boot only. In this case, the BIOS uses the same boot configuration for both default and one-time boot configuration.<br>} |
| 1 | enum8 | **OrderAndFailThroughMode**<br>Possible values:<br>{<br>UnorderedAndLimitedFailThrough (0x00) – The boot sources specified in the array below can be applied in any order and in the case of failure to boot from the boot sources specified in the array, other boot sources that are specified as possible boot sources for this attribute in the BIOS Attribute Table shall not be tried.<br>UnorderedAndFailThrough (0x01) – The boot sources specified in the array below can be applied in any order and in the case of failure to boot from the boot sources specified in the array, other boot sources that are specified as possible boot sources for this attribute in the BIOS Attribute Table can be tried in any order.<br>OrderedAndLimitedFailThrough (0x02) – The boot sources specified in the array below must be applied in the order specified in the array and in the case of failure to boot from the boot sources specified in the array, other boot sources that are specified as possible boot sources for this attribute in the BIOS Attribute Table shall not be tried.<br>OrderedAndFailThrough (0x03) – The boot sources specified in the array below must be applied in the order specified in the array and in the case of failure to boot from the boot sources specified in the array, other boot sources that are specified as possible boot sources for this attribute in the BIOS Attribute Table can be tried in any order.<br>} |
| 2 | uint8 | **NumberOfPendingBootSourceSettings (N)**<br>Specifies the number of boot source settings that are in the pending boot configuration |

| Byte | Type | Field |
|---|---|---|
| 3 | uint8 | **BootSourceStringHandleIndex[0]**<br>An index into the array of the possible values of string handles for the first boot source setting value. This index points to an entry in the array of string handles representing possible values for this attribute provided in the BIOS Attribute Table. |
| 4 | uint8 | **BootSourceStringHandleIndex[1]**<br>An index into the array of the possible values of string handles for the second boot source setting value. This index points to an entry in the array of string handles representing possible values for this attribute provided in the BIOS Attribute Table. |
| … | … | **…** |
| N+2 | uint8 | **BootSourceStringHandleIndex[N-1]**<br>An index into the array of the possible values of string handles for the Nth boot source setting value. This index points to an entry in the array of string handles representing possible values for this attribute provided in the BIOS Attribute Table. |

The string handle index in Table 28 is an 8-bit index into the array (provided in the BIOS Attribute Table) of the possible boot source string handles for this attribute.

The BIOS Attribute Pending Value Table can contain changes to the settings for multiple boot configurations. When multiple BIOS attributes of type BIOSBootConfigSetting are provided in the BIOS Attribute Pending Value Table, the following rules apply:

- At most one BIOS attribute shall exist among the BIOS attributes of type BIOSBootConfigSetting with BootConfigType set to Default, DefaultAndNext, or DefaultAndOnetime. In other words, the BIOS Attribute Pending Value Table shall contain at most one default boot configuration.
- At most one BIOS attribute shall exist among the BIOS attributes of type BIOSBootConfigSetting with BootConfigType set to Next or DefaultAndNext. In other words, the BIOS Attribute Pending Value Table shall contain at most one next boot configuration.
- At most one BIOS attribute shall exist among the BIOS attributes of type BIOSBootConfigSetting with BootConfigType set to Onetime or DefaultAndOnetime. In other words, the BIOS Attribute Pending Value Table shall contain at most one onetime boot configuration.
- For the next boot, the boot configuration with BootConfigType set to Onetime or DefaultAndOnetime takes precedence over the boot configuration with BootConfigType set to Next or DefaultAndNext.

The specific fields for the BIOSCollection and BIOSCollectionReadOnly types are described in Table 29.

**Table 29 – Specific BIOS Attribute Pending Value Table Fields for BIOSCollection and BIOSCollectionReadOnly Types**

| Byte | Type | Field |
|---|---|---|
| 0 | uint8 | **NumberOfAttributes (N)**<br>The number of BIOS attributes that belong to the pending value of the BIOS collection |
| 1:2 | uintt16 | **AttributeHandle[0]**<br>A handle that is used to identify the first attribute in the pending value of the BIOS collection. This handle points to an attribute in the BIOS Attribute Table. |

| Byte | Type | Field |
|---|---|---|
| .. | uint16 | **….** |
| 2N-1:2N | uint16 | **AttributeHandle[N-1]**<br>A handle that is used to identify the Nth attribute in the pending value of the BIOS collection. This handle points to an attribute in the BIOS Attribute Table. |

The BIOS Attribute Pending Value Table shall not contain a BIOS attribute of type BIOSCollection or BIOSCollectionReadOnly with CollectionType set to UnorderedAndUnmodifiable or OrderedAndUnmodifiable. In other words, the BIOS Attribute Pending Value Table shall not contain an unmodifiable BIOS collection.

The specific fields for the BIOSConfigSet are described in Table 30. The BIOS Attribute Pending Value Table shall contain at most one attribute of type BIOSConfigSet.

**Table 30 – Specific BIOS Attribute Pending Value Table Fields for the BIOSConfigSet Type**

| Byte | Type | Field |
|---|---|---|
| 0 | uint8 | **ConfigSetStringHandleIndex**<br>An index into the array of the possible values of BIOS configuration sets. This index points to an entry in the array of string handles representing possible values for this attribute provided in the BIOS Attribute Table. |

# 8 PLDM Commands for BIOS Control and Configuration

The PLDM commands for BIOS Control and Configuration are defined in this section.

Table 31 defines the PLDM command codes for the PLDM for BIOS Control and Configuration.

**Table 31 – PLDM for BIOS Control and Configuration Command Codes**

| Command | Code Value | Requirement | Section |
|---|---|---|---|
| GetBIOSTable | 0x01 | Mandatory | 8.1 |
| SetBIOSTable | 0x02 | Mandatory | 8.2 |
| UpdateBIOSTable | 0x03 | Optional | 8.3 |
| GetBIOSTableTags | 0x04 | Optional | 8.4 |
| SetBIOSTableTags | 0x05 | Optional | 8.5 |
| AcceptBIOSAttributesPendingValues | 0x06 | Mandatory | 8.6 |
| SetBIOSAttributeCurrentValue | 0x07 | Optional | 8.7 |
| GetBIOSAttributeCurrentValueByHandle | 0x08 | Optional | 8.8 |
| GetBIOSAttributePendingValueByHandle | 0x09 | Optional | 8.9 |
| GetBIOSAttributeCurrentValueByType | 0x0a | Optional | 8.10 |
| GetBIOSAttributePendingValueByType | 0x0b | Optional | 8.11 |
| GetDateTime | 0x0c | Conditional[1] | 8.12 |

| Command | Code Value | Requirement | Section |
|---|---|---|---|
| SetDateTime | 0x0d | Conditional[1] | 8.13 |
| GetBIOSStringTableStringType | 0x0e | Optional | 8.14 |
| SetBIOSStringTableStringType | 0x0f | Optional | 8.15 |
| [1]These commands are optional, but if SetDateTime is implemented, then GetDateTime must be implemented. | | | |

The requirements specified in Table 31 are relative to the services provided by the PLDM terminus. A compliant implementation must support at least one type of BIOS attribute in the BIOS tables.

## 8.1 GetBIOSTable

The GetBIOSTable command, described in Table 32, is used by the MC (or BIOS) to get a table from the BIOS (or MC) using one or more PLDM requests. For multipart transfers, see 9.1.

**Table 32 – GetBIOSTable Command**

| Byte | Type | Request Data |
|---|---|---|
| 0:3 | uint32 | **DataTransferHandle**<br>A handle that is used to identify a BIOS table transfer. This handle is ignored by the responder when the TransferOperationFlag is set to GetFirstPart. |
| 4 | enum8 | **TransferOperationFlag**<br>The transfer operation flag that indicates whether this is the start of a multipart transfer<br>Possible values: {GetNextPart=0x00, GetFirstPart=0x01} |
| 5 | enum8 | **TableType**<br>Indicates what table is being transferred<br>Possible values:<br>{<br>BIOSStringTable=0x0,<br>BIOSAttributeTable=0x1,<br>BIOSAttributeValueTable=0x2,<br>BIOSAttributePendingValueTable=0x3<br>} |
| **Byte** | **Type** | **Response Data** |
| 0 | enum8 | **CompletionCode**<br>Possible values:<br>{<br>PLDM_BASE_CODES,<br>INVALID_DATA_TRANSFER_HANDLE=0x80,<br>INVALID_TRANSFER_OPERATION_FLAG=0x81,<br>BIOS_TABLE_UNAVAILABLE=0x83,<br>INVALID_BIOS_TABLE_DATA_INTEGRITY_CHECK=0x84,<br>INVALID_BIOS_TABLE_TYPE=0x85<br>} |
| 1:4 | uint32 | **NextDataTransferHandle**<br>A handle that is used to identify the next portion of the transfer |

| Byte | Type | Response Data |
|---|---|---|
| 5 | enum8 | **TransferFlag**<br>The transfer flag that indicates what part of the transfer this response represents<br>Possible values: {Start = 0x1, Middle = 0x2, End = 0x4, StartAndEnd = 0x5} |
| Variable | - | **TableData**<br>Table type specific data. See the data structures in Section 7. |

## 8.2 SetBIOSTable

The SetBIOSTable command, described in Table 33, is used by the BIOS (or MC) to set a BIOS table on the MC using one or more PLDM requests. For multipart transfers, see 9.1.

**Table 33 – SetBIOSTable Command**

| Byte | Type | Request Data |
|---|---|---|
| 0:3 | uint32 | **DataTransferHandle**<br>A handle that is used to identify a BIOS table transfer. This handle is ignored by the responder when the TransferFlag is set to Start or StartAndEnd. |
| 4 | enum8 | **TransferFlag**<br>The transfer flag that indicates what part of the transfer this request represents<br>Possible values: {Start = 0x1, Middle = 0x2, End = 0x4, StartAndEnd = 0x5} |
| 5 | enum8 | **TableType**<br>Indicates what table is being transferred<br>Possible values:<br>{BIOSStringTable=0x0, BIOSAttributeTable=0x1, BIOSAttributeValueTable=0x2} |
| Variable | – | **TableData**<br>Table type specific data. See the data structures in 8.1. |
| **Byte** | **Type** | **Response Data** |
| 0 | enum8 | **CompletionCode**<br>Possible values:<br>{<br>PLDM_BASE_CODES,<br>INVALID_DATA_TRANSFER_HANDLE=0x80,<br>INVALID_TRANSFER_FLAG=0x82,<br>INVALID_BIOS_TABLE_DATA_INTEGRITY_CHECK=0x84,<br>INVALID_BIOS_TABLE_TYPE=0x85<br>} |
| 1:4 | uint32 | **NextDataTransferHandle**<br>A handle that is used to identify the next portion of the transfer |

## 8.3 UpdateBIOSTable

The UpdateBIOSTable command, described in Table 34, is used by the BIOS (or MC) to update a BIOS table using one or more PLDM requests. This action involves updating the existing entries, adding new entries, or both. For multipart transfers, see 9.1. The BIOS (or MC) also provides the integrity checksum for the data transferred in this update.

**Table 34 – UpdateBIOSTable Command**

| Byte | Type | Request Data |
|---|---|---|
| 0:3 | uint32 | **DataTransferHandle**<br>A handle that is used to identify a BIOS table transfer. This handle is ignored by the responder when the TransferFlag is set to Start or StartAndEnd. |
| 4 | enum8 | **TransferFlag**<br>The transfer flag that indicates what part of the transfer this request represents<br>Possible values: {Start = 0x1, Middle = 0x2, End = 0x4, StartAndEnd = 0x5} |
| 5 | enum8 | **TableType**<br>Indicates what table is being transferred<br>Possible values:<br>{BIOSStringTable=0x0, BIOSAttributeTable=0x1, BIOSAttributeValueTable=0x2} |
| Variable | - | **TableData**<br>Table type specific data. See the data structures in the previous section. |
| **Byte** | **Type** | **Response Data** |
| 0 | enum8 | **CompletionCode**<br>Possible values:<br>{<br>PLDM_BASE_CODES,<br>INVALID_DATA_TRANSFER_HANDLE=0x80,<br>INVALID_TRANSFER_FLAG=0x82,<br>INVALID_BIOS_TABLE_DATA_INTEGRITY_CHECK=0x84,<br>INVALID_BIOS_TABLE_TYPE=0x85<br>} |
| 1:4 | uint32 | **NextDataTransferHandle**<br>A handle that is used to identify the next portion of the transfer |

## 8.4 GetBIOSTableTags

The GetBiosTableTags command, described in Table 35, is used by the BIOS to query the tags of the BIOS tables maintained by the MC. The use of BIOS table tags is described in 6.6.

**Table 35 – GetBIOSTableTags Command**

| Byte | Type | Request Data |
|---|---|---|
| 0 | uint8 | **NumberOfTables (N)** |
| 1 | enum8 | **TableType[0]**<br>Possible values:<br>{BIOSStringTable=0x0, BIOSAttributeTable=0x1, BIOSAttributeValueTable=0x2} |
| 2 | enum8 | **….** |
| N | enum8 | **TableType[N-1]**<br>Possible values:<br>{BIOSStringTable=0x0, BIOSAttributeTable=0x1, BIOSAttributeValueTable=0x2} |
| **Byte** | **Type** | **Response Data** |
| 0 | enum8 | **CompletionCode**<br>Possible values:<br>{<br>PLDM_BASE_CODES,<br>BIOS_TABLE_TAG_UNAVAILABLE=0x86,<br>INVALID_BIOS_TABLE_TAG_TYPE=0x87<br>} |
| 1:4 | uint32 | **TableTag[0]** |
| … | uint32 | **….** |
| 4N-3:4N | uint32 | **TableTag[N-1]** |

## 8.5 SetBIOSTableTags

The SetBIOSTableTags command, described in Table 36, is used by the BIOS to set the tags of the tables that the MC maintains. The BIOS can use this command after updating the BIOS tables. The use of BIOS table tags is described in 6.6.

A table tag for a particular table type shall be specified at most one time in the request data. If multiple table tags for a particular table type are specified in the request, the responder may return an error or may process multiple tags in an order that is implementation specific.

**Table 36 – SetBIOSTableTags Command**

| Byte | Type | Request Data |
|---|---|---|
| 0 | uint8 | **NumberOfTables (N)** |
| 1 | enum8 | **TableType[0]**<br>Possible values:<br>{BIOSStringTable=0x0, BIOSAttributeTable=0x1, BIOSAttributeValueTable=0x2} |
| 2:5 | uint32 | **TableTag[0]** |
| … | … | **….** |
| 5*N-4 | enum8 | **TableType[N-1]**<br>Possible values:<br>{BIOSStringTable=0x0, BIOSAttributeTable=0x1, BIOSAttributeValueTable=0x2} |
| 5*N-3:5*N | uint32 | **TableTag[N-1]** |

| Byte | Type | Response Data |
|---|---|---|
| 0 | enum8 | **CompletionCode**<br>Possible values :<br>{<br>PLDM_BASE_CODES,<br>INVALID_BIOS_TABLE_TAG_TYPE=0x87<br>} |

## 8.6 AcceptBIOSAttributesPendingValues

The AcceptBIOSAttributesPendingValues command, described in Table 37, is used by the BIOS to signal the MC of the acceptance of the pending values of the BIOS attributes. The pending values of the attributes not specified in these transfers are rejected by the BIOS. The MC must clear the pending table after processing this command.

**Table 37 – AcceptBIOSAttributesPendingValues Command**

| Byte | Type | Request Data |
|---|---|---|
| 0:3 | uint32 | **DataTransferHandle**<br>A handle that is used to identify this transfer. This handle is ignored by the responder when the TransferFlag is set to Start or StartAndEnd. |
| 4 | enum8 | **TransferFlag**<br>The transfer flag that indicates what part of the transfer this request represents<br>Possible values: {Start=0x1, Middle=0x2, End=0x4, StartAndEnd = 0x5} |
| Variable | - | **BIOSAttributesHandles (see Table 38)** |
| **Byte** | **Type** | **Response Data** |
| 0 | enum8 | **CompletionCode**<br>Possible values :<br>{<br>PLDM_BASE_CODES,<br>INVALID_DATA_TRANSFER_HANDLE=0x80,<br>INVALID_TRANSFER_FLAG=0x82,<br>INVALID_BIOS_ATTR_HANDLE=0x88<br>} |
| 1:4 | uint32 | **NextDataTransferHandle**<br>A handle that is used to identify the next portion of the transfer |

The array of BIOS attribute handles at the PLDM level is represented as shown in Table 38. This data structure is transferred using the PLDM command described in this section. For more details on multipart data transfers, see 9.1.

**Table 38 – PLDM Representation of BIOSAttributesHandles**

| Byte | Type | Field |
|---|---|---|
| 0-1 | uint16 | **NumberOfAttributeHandles**<br>The total number of attribute handles present in this structure. This represents the number of accepted pending attribute changes. If the attribute handle for a pending attribute value is not present in this table, the pending value for that attribute shall be rejected and discarded. If all the pending attribute values are rejected, the NumberOfAttributeHandles shall be set to 0. |
| 2-3 | uint16 | **AttributeHandle[0]**<br>A handle that is used to identify the first attribute in this structure. This handle points to an attribute in the BIOS Attribute Table. |
| … | … | **…** |
| … | uint16 | **AttributeHandle[N-1]**<br>A handle that is used to identify the Nth attribute in this structure. This handle points to an attribute in the BIOS Attribute Table. |
| Variable | … | **Pad**<br>0 or 2 pad bytes. The value stored in each pad byte is 0x00.<br>The number of pad bytes can be calculated as follows:<br>If (NumberOfAttributeHandles  modulo  2 == 1) then NumberOfPadBytes = 0;<br>else NumberOfPadBytes = 2; |
| … | uint32 | **BIOSAttributesHandlesIntegrityChecksum**<br>Integrity checksum on the BIOS attribute handles shown above including the NumberOfAttributesHandles field and the pad information.<br>For this specification, the CRC-32 algorithm with the polynomial $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1$ (same as the one used by IEEE 802.3) must be used for the integrity checksum computation. The CRC computation involves processing a byte at a time with the least significant bit first. |

## 8.7 SetBIOSAttributeCurrentValue

The SetBIOSAttributeCurrentValue command, described in Table 39, is used by the BIOS (or MC) to set the current value of a BIOS attribute on the MC using one or more PLDM requests. For multipart transfers, see 9.1.

**Table 39 – SetBIOSAttributeCurrentValue Command**

| Byte | Type | Request Data |
|---|---|---|
| 0:3 | uint32 | **DataTransferHandle**<br>A handle that is used to identify a BIOS table transfer. This handle is ignored by the responder when the TransferFlag is set to Start or StartAndEnd. |
| 4 | enum8 | **TransferFlag**<br>The transfer flag that indicates what part of the transfer this request represents<br>Possible values: {Start=0x1, Middle=0x2, End=0x4, StartAndEnd = 0x5} |
| Variable | … | **AttributeData**<br>See Table 14 through Table 21 for the format of data. For this command, the AttributeData contains exactly one attribute whose current value is being set. |

| Byte | Type | Response Data |
|---|---|---|
| 0 | enum8 | **CompletionCode**<br>Possible values :<br>{<br>PLDM_BASE_CODES,<br>INVALID_DATA_TRANSFER_HANDLE=0x80,<br>INVALID_TRANSFER_FLAG=0x82,<br>} |
| 1:4 | uint32 | **NextDataTransferHandle**<br>A handle that is used to identify the next portion of the transfer |

## 8.8 GetBIOSAttributeCurrentValueByHandle

The GetBIOSAttributeCurrentValueByHandle command, described in Table 40, is used by the BIOS (or MC) to get the current value of a BIOS attribute (identified by an AttributeHandle) from the MC (or BIOS) using one or more PLDM requests. For multipart transfers, see 9.1.

**Table 40 – GetBIOSAttributeCurrentValueByHandle Command**

| Byte | Type | Request Data |
|---|---|---|
| 0:3 | uint32 | **DataTransferHandle**<br>A handle that is used to identify a BIOS attribute transfer. This handle is ignored by the responder when the TransferOperationFlag is set to GetFirstPart. |
| 4 | enum8 | **TransferOperationFlag**<br>The operation flag that indicates whether this is the start of the transfer<br>Possible values: {GetNextPart=0x00, GetFirstPart=0x01} |
| 5:6 | uint16 | **AttributeHandle**<br>A handle that is used to identify the BIOS attribute |
| **Byte** | **Type** | **Response Data** |
| 0 | enum8 | **CompletionCode**<br>Possible values:<br>{<br>PLDM_BASE_CODES,<br>INVALID_DATA_TRANSFER_HANDLE=0x80,<br>INVALID_TRANSFER_OPERATION_FLAG=0x81,<br>INVALID_BIOS_ATTR_HANDLE=0x88<br>} |
| 1:4 | uint32 | **NextDataTransferHandle**<br>A handle that is used to identify the next portion of the transfer |
| 5 | enum8 | **TransferFlag**<br>The transfer flag that indicates what part of the transfer this response represents<br>Possible values: {Start=0x1, Middle=0x2, End=0x4, StartAndEnd = 0x5} |
| Variable | … | **AttributeData**<br>See Table 14 through Table 21 for the format of data. For this command, the AttributeData contains exactly one attribute whose current value is being provided. |

## 8.9 GetBIOSAttributePendingValueByHandle

The GetBIOSAttributePendingValueByHandle command, described in Table 41, is used by the BIOS (or MC) to get the pending value of a BIOS attribute (identified by an AttributeHandle) from the MC using one or more PLDM requests. For multipart transfers, see 9.1.

**Table 41 – GetBIOSAttributePendingValueByHandle**

| Byte | Type | Request Data |
|---|---|---|
| 0:3 | uint32 | **DataTransferHandle**<br>A handle that is used to identify a BIOS attribute transfer. This handle is ignored by the responder when the TransferOperationFlag is set to GetFirstPart. |
| 4 | enum8 | **TransferOperationFlag**<br>The transfer operation flag that indicates whether this is the start of the transfer<br>Possible values: {GetNextPart=0x00, GetFirstPart=0x01} |
| 5:6 | uint16 | **BIOSAttributeHandle**<br>A handle that is used to identify the BIOS attribute |
| **Byte** | **Type** | **Response Data** |
| 0 | enum8 | **CompletionCode**<br>Possible values:<br>{<br>PLDM_BASE_CODES,<br>INVALID_DATA_TRANSFER_HANDLE=0x80,<br>INVALID_TRANSFER_OPERATION_FLAG=0x81,<br>INVALID_BIOS_ATTR_HANDLE=0x88<br>} |
| 1:4 | uint32 | **NextDataTransferHandle**<br>A handle that is used to identify the next portion of the transfer |
| 5 | enum8 | **TransferFlag**<br>The transfer flag that indicates what part of the transfer this response represents<br>Possible values: {Start=0x1, Middle=0x2, End=0x4, StartAndEnd = 0x5} |
| Variable | … | **AttributeData**<br>See Table 23 through Table 30 for the format of data. For this command, the AttributeData contains exactly one attribute whose pending value is being provided. |

## 8.10 GetBIOSAttributeCurrentValueByType

The GetBIOSAttributeCurrentValueByType command, described in Table 42, is used by the BIOS (or MC) to get the current values of BIOS attributes of a specific AttributeType from the MC (or BIOS) using one or more PLDM requests. For multipart transfers, see 9.1.

**Table 42 – GetBIOSAttributeCurrentValueByType**

| Byte | Type | Request Data |
|---|---|---|
| 0:3 | uint32 | **DataTransferHandle**<br>A handle that is used to identify a BIOS attribute transfer. This handle is ignored by the responder when the TransferOperationFlag is set to GetFirstPart. |
| 4 | enum8 | **TransferOperationFlag**<br>The transfer operation flag that indicates whether this is the start of the transfer<br>Possible values: {GetNextPart=0x00, GetFirstPart=0x01} |
| 5 | enum8 | **AttributeType**<br>The type of the BIOS attribute(s). Refer to Table 14 for the possible values of attribute type. |
| **Byte** | **Type** | **Response Data** |
| 0 | enum8 | **CompletionCode**<br>Possible values:<br>{<br>PLDM_BASE_CODES,<br>INVALID_DATA_TRANSFER_HANDLE=0x80,<br>INVALID_TRANSFER_OPERATION_FLAG=0x81,<br>INVALID_BIOS_ATTR_TYPE=0x89<br>} |
| 1:4 | uint32 | **NextDataTransferHandle**<br>A handle that is used to identify the next portion of the transfer |
| 5 | enum8 | **TransferFlag**<br>The transfer flag that indicates what part of the transfer this response represents<br>Possible values: {Start=0x1, Middle=0x2, End=0x4, StartAndEnd = 0x5} |
| Variable | … | **AttributeData**<br>See Table 14 through Table 21 for the format of data. For this command, the AttributeData contains the current values of all the BIOS attributes of the type specified in the AttributeType field in the request data. |

## 8.11 GetBIOSAttributePendingValueByType

The GetBIOSAttributePendingValueByType command, described in Table 43, is used by the BIOS (or MC) to get the pending values of BIOS attributes of a specific AttributeType from the MC (or BIOS) using one or more PLDM requests. For multipart transfers, see 9.1.

**Table 43 – GetBIOSAttributePendingValueByType Command**

| **Byte** | **Type** | **Request Data** |
|---|---|---|
| 0:3 | uint32 | **DataTransferHandle**<br>A handle that is used to identify a BIOS attribute transfer. This handle is ignored by the responder when the TransferOperationFlag is set to GetFirstPart. |
| 4 | enum8 | **TransferOperationFlag**<br>The transfer operation flag that indicates whether this is the start of the transfer<br>Possible values: {GetNextPart=0x00, GetFirstPart=0x01} |
| 5 | enum8 | **AttributeType**<br>The type of the BIOS attribute(s). Refer to Table 23 for the possible values of attribute type. |
| **Byte** | **Type** | **Response Data** |
| 0 | enum8 | **CompletionCode**<br>Possible values:<br>{<br>PLDM_BASE_CODES,<br>INVALID_DATA_TRANSFER_HANDLE=0x80,<br>INVALID_TRANSFER_OPERATION_FLAG=0x81,<br>INVALID_BIOS_ATTR_TYPE=0x89<br>} |
| 1:4 | uint32 | **NextDataTransferHandle**<br>A handle that is used to identify the next portion of the transfer |
| 5 | enum8 | **TransferFlag**<br>The transfer flag that indicates what part of the transfer this response represents<br>Possible values: {Start=0x1, Middle=0x2, End=0x4, StartAndEnd = 0x5} |
| Variable | … | **AttributeData**<br>See Table 23 through Table 30 for the format of data. For this command, the AttributeData contains the current values of all the BIOS attributes of the type specified in the AttributeType field in the request data. |

## 8.12 GetDateTime

The GetDateTime command, described in Table 44, is used to get the date and time information.

**Table 44 – GetDateTime Command**

| **Byte** | **Type** | **Request Data** |
|---|---|---|
| – | – | None |
| **Byte** | **Type** | **Response Data** |
| 0 | enum8 | **CompletionCode**<br>Possible values:<br>{<br>PLDM_BASE_CODES,<br>NO_DATE_TIME_INFO_AVAILABLE=0x8A<br>} |
| 1 | uint8 | **Seconds in BCD format – Valid range of values [0, 59]** |
| 2 | uint8 | **Minutes in BCD format – Valid range of values [0, 59]** |
| 3 | uint8 | **Hours in BCD format – Valid range of values [0, 23]** |
| 4 | uint8 | **Day of the Month in BCD format – Valid range of values [1, 31]** |
| 5 | uint8 | **Month in BCD format – Valid range of values [1, 12]** |
| 6:7 | uint16 | **Year in BCD format (4 digits)** |

## 8.13 SetDateTime

The SetDateTime command, described in Table 45, is used to set the date and time information.

**Table 45 – SetDateTime Command**

| **Byte** | **Type** | **Request Data** |
|---|---|---|
| 0 | uint8 | **Seconds in BCD format – Valid range of values [0, 59]** |
| 1 | uint8 | **Minutes in BCD format – Valid range of values [0, 59]** |
| 2 | uint8 | **Hours in BCD format – Valid range of values [0, 23]** |
| 3 | uint8 | **Day of the Month in BCD format – Valid range of values [1, 31]** |
| 4 | uint8 | **Month in BCD format – Valid range of values [1, 12]** |
| 5:6 | uint16 | **Year in BCD format (4 digits)** |
| **Byte** | **Type** | **Response Data** |
| 0 | enum8 | **CompletionCode**<br>Possible values:<br>{<br>PLDM_BASE_CODES<br>} |

## 8.14 GetBIOSStringTableStringType

The GetBIOSStringTableStringType command is used to get the type of strings used in the BIOS String Table.

**Table 46 – GetBIOSStringTableStringType Command**

| Byte | Type | Request Data |
|---|---|---|
| --- | --- | None |
| **Byte** | **Type** | **Response Data** |
| 0 | enum8 | **CompletionCode**<br>Possible values:<br>{<br>PLDM_BASE_CODES<br>} |
| 1 | enum8 | **StringType**<br>The type of strings used in the BIOS String Table<br>Possible values:<br>{Unknown=0x00, ASCII=0x01, Hex=0x02, UTF-8=0x03, UTF-16LE=0x04, UTF-16BE=0x05, Vendor Specific=0xFF} |

## 8.15 SetBIOSStringTableStringType

The SetBIOSStringTableStringType command is used to set the type of strings used in the BIOS String Table.

**Table 47 – SetBIOSStringTableStringType Command**

| Byte | Type | Request Data |
|---|---|---|
| 0 | enum8 | **StringType**<br>The type of strings used in the BIOS String Table.<br>Possible values:<br>{Unknown=0x00, ASCII=0x01, Hex=0x02, UTF-8=0x03, UTF-16LE=0x04, UTF-16BE=0x05, Vendor Specific=0xFF} |
| **Byte** | **Type** | **Response Data** |
| 0 | enum8 | **CompletionCode**<br>Possible values:<br>{<br>PLDM_BASE_CODES,<br>INVALID_STRING_TYPE=0x8B<br>} |

## 8.16 PLDM for BIOS Control and Configuration Version

The version of this PLDM for BIOS Control and Configuration specification shall be 1.0.0 (major version number 1, minor version number 0, update version number 0, and no alpha version).

For the GetPLDMVersion command described in DSP0240, the version of this specification is reported using the encoding as: `0xF1F0F000.`

# 9 BIOS/MC PLDM Communications Examples

The previous two sections described the data structures and commands that the BIOS and the MC can use to communicate the information about the BIOS attributes. This section provides some examples of PLDM communications using the PLDM commands defined in this specification.

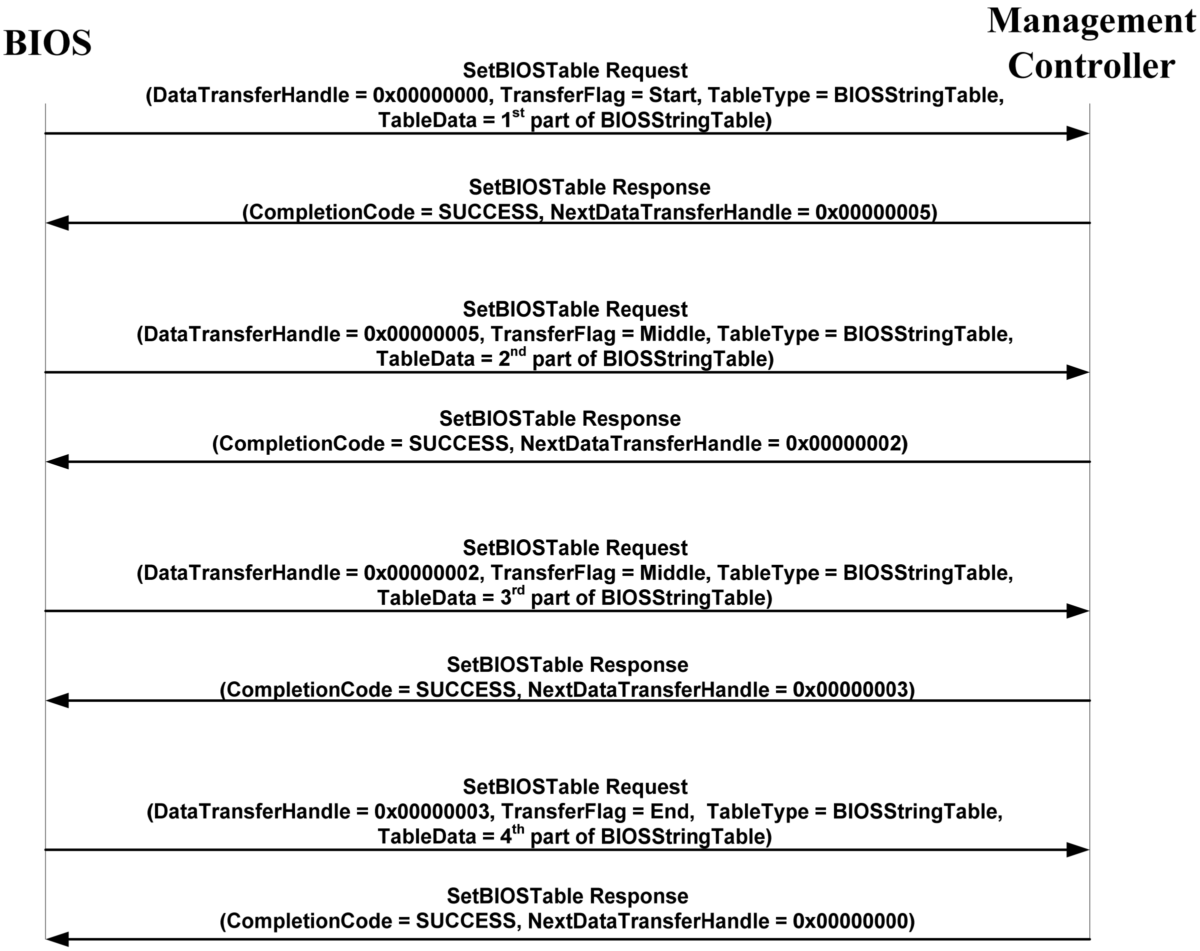
## 9.1 Multipart Transfers

The commands defined in section 8 for transferring BIOS table data or attribute data support multipart transfers. The Get* and Set* commands use flags and data transfer handles to perform multipart transfers. Following are some requirements for using TransferOperationFlag, TransferFlag, and DataTransferHandle for a given data transfer:

- For initiating a data transfer (or getting the first part of data) using a Get* command, the TransferOperationFlag shall be set to GetFirstPart in the request of the Get* command.
- For transferring a part other than the first part of data using a Get* command, the TransferOperationFlag shall be set to GetNextPart and the DataTransferHandle shall be set to the NextDataTransferHandle that was obtained in the response of the previous Get* command for this data transfer.
- The TransferFlag specified in the request of a Set* command or the response of a Get* command has the following meanings:
  - – Start, which is the first part of the data transfer
  - – Middle, which is neither the first nor the last part of the data transfer
  - – End, which is the last part of the data transfer
  - – StartAndEnd, which is the first and the last part of the data transfer
- The requester shall consider a data transfer complete when the TransferFlag in the response of a Get* command is set to End or StartAndEnd.
- The responder shall consider a data transfer complete when the TransferFlag in the request of a Set* command is set to End or StartAndEnd.

The following two examples show how multipart transfers can be performed using the generic mechanism defined in the commands.

731 EXAMPLE 1: In this example, the BIOS is transferring the BIOS String Table to the MC using the SetBIOSTable
732 command. The transfer is divided into four parts. Figure 1 shows the flow of the data transfer.


BIOS
Management Controller
SetBIOSTable Request
(DataTransferHandle = 0x00000000, TransferFlag = Start, TableType = BIOSStringTable, TableData = 1st part of BIOSStringTable)
SetBIOSTable Response
(CompletionCode = SUCCESS, NextDataTransferHandle = 0x00000005)
SetBIOSTable Request
(DataTransferHandle = 0x00000005, TransferFlag = Middle, TableType = BIOSStringTable, TableData = 2nd part of BIOSStringTable)
SetBIOSTable Response
(CompletionCode = SUCCESS, NextDataTransferHandle = 0x00000002)
SetBIOSTable Request
(DataTransferHandle = 0x00000002, TransferFlag = Middle, TableType = BIOSStringTable, TableData = 3rd part of BIOSStringTable)
SetBIOSTable Response
(CompletionCode = SUCCESS, NextDataTransferHandle = 0x00000003)
SetBIOSTable Request
(DataTransferHandle = 0x00000003, TransferFlag = End, TableType = BIOSStringTable, TableData = 4th part of BIOSStringTable)
SetBIOSTable Response
(CompletionCode = SUCCESS, NextDataTransferHandle = 0x00000000)

733

734 **Figure 1 – Multipart BIOS Table Transfer Using the SetBIOSTable Command**

EXAMPLE 2: In this example, the BIOS is transferring the BIOS Attribute Pending Value Table from the MC using the GetBIOSTable command. The transfer is divided into three parts. Figure 2 shows the flow of the data transfer.

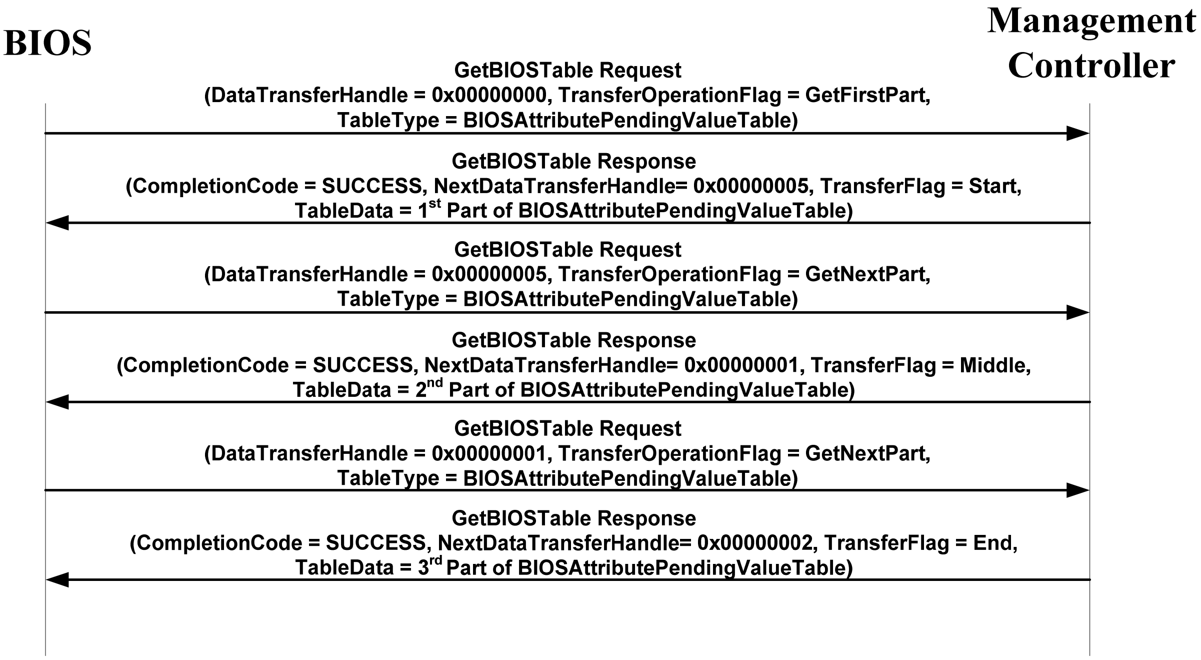

BIOS
Management Controller
GetBIOSTable Request (DataTransferHandle = 0x00000000, TransferOperationFlag = GetFirstPart, TableType = BIOSAttributePendingValueTable)
GetBIOSTable Response (CompletionCode = SUCCESS, NextDataTransferHandle= 0x00000005, TransferFlag = Start, TableData = 1st Part of BIOSAttributePendingValueTable)
GetBIOSTable Request (DataTransferHandle = 0x00000005, TransferOperationFlag = GetNextPart, TableType = BIOSAttributePendingValueTable)
GetBIOSTable Response (CompletionCode = SUCCESS, NextDataTransferHandle= 0x00000001, TransferFlag = Middle, TableData = 2nd Part of BIOSAttributePendingValueTable)
GetBIOSTable Request (DataTransferHandle = 0x00000001, TransferOperationFlag = GetNextPart, TableType = BIOSAttributePendingValueTable)
GetBIOSTable Response (CompletionCode = SUCCESS, NextDataTransferHandle= 0x00000002, TransferFlag = End, TableData = 3rd Part of BIOSAttributePendingValueTable)

**Figure 2 – Multipart BIOS Table Transfer Using the GetBIOSTable Command**

## 9.2 BIOS Table Initialization on MC

In Figure 3, the BIOS sets the BIOS tables for the first time on the MC. No authentication of any entities occurs in this initialization example. The BIOS first queries the BIOS table tags using the GetBIOSTableTags command. The response from the MC to this command indicates that the MC does not have any BIOS table tags. Upon finding that the MC does not have the BIOS table tags, the BIOS initializes the BIOS tables (BIOS String Table, BIOS Attribute Table, and BIOS Attribute Value Table, in that order) on the MC by using SetBIOSTable command. At the end of the initialization, the BIOS sets up the BIOS table tags on the MC by using SetBIOSTableTags command.

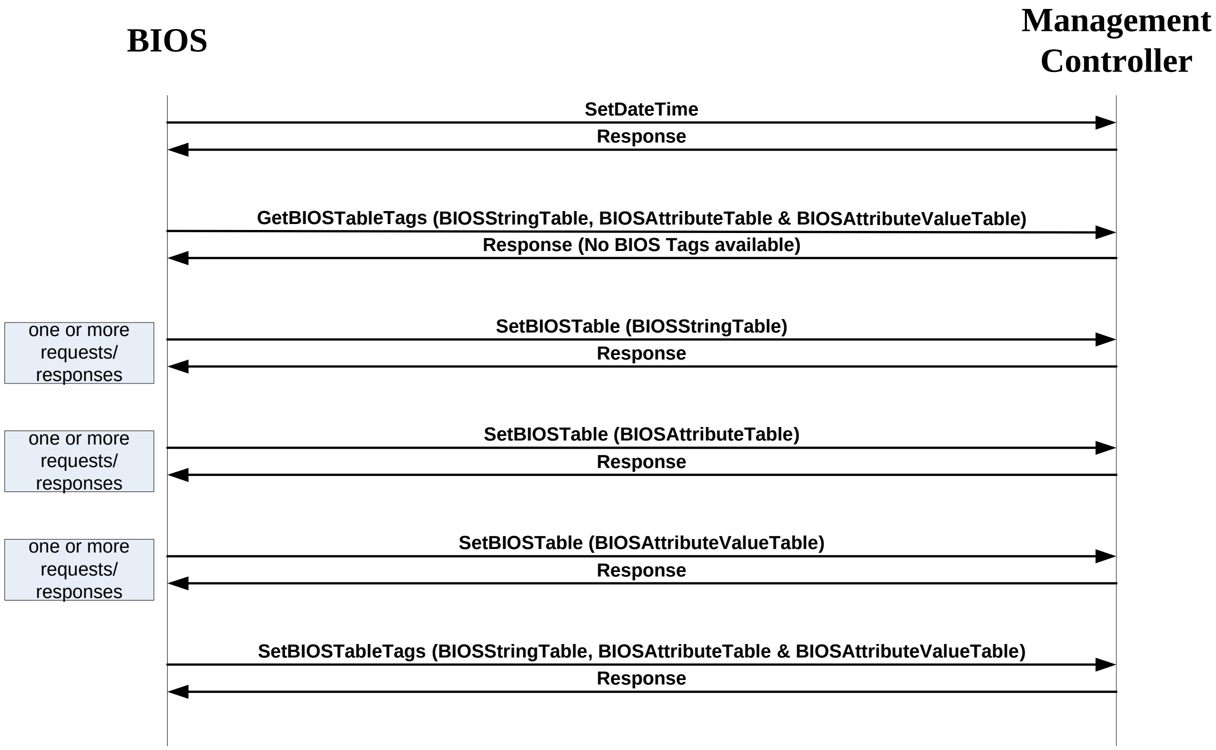

BIOS
Management Controller
SetDateTime
Response
GetBIOSTableTags (BIOSStringTable, BIOSAttributeTable & BIOSAttributeValueTable)
Response (No BIOS Tags available)
one or more requests/ responses
SetBIOSTable (BIOSStringTable)
Response
one or more requests/ responses
SetBIOSTable (BIOSAttributeTable)
Response
one or more requests/ responses
SetBIOSTable (BIOSAttributeValueTable)
Response
SetBIOSTableTags (BIOSStringTable, BIOSAttributeTable & BIOSAttributeValueTable)
Response

**Figure 3 – Example of BIOS Table Initialization**

## 9.3 No BIOS Setting Changes

In this example, the BIOS settings have not changed remotely or locally and the MC has the latest copies of the BIOS tables. The two variations of this example are as follows:

1) The BIOS does not authenticate the MC before the BIOS/MC communications.

2) The BIOS authenticates the MC before the BIOS/MC communications.

Figure 4 shows BIOS/MC communications without MC authentication. The BIOS queries the MC using the GetBIOSTableTags command. The MC responds with the BIOS table tags for the BIOS String Table, BIOS Attribute Table, and BIOS Attribute Value Table. Based on the response from the MC, the BIOS determines that the MC has the latest copies of the BIOS tables. The BIOS then sends the GetBIOSTable command to the MC to get the BIOS Attribute Pending Value Table. Because no remote changes occurred to the BIOS attributes, the MC response to the BIOS indicates no pending values. At this point, the BIOS knows that no remote changes were made to the BIOS attributes.

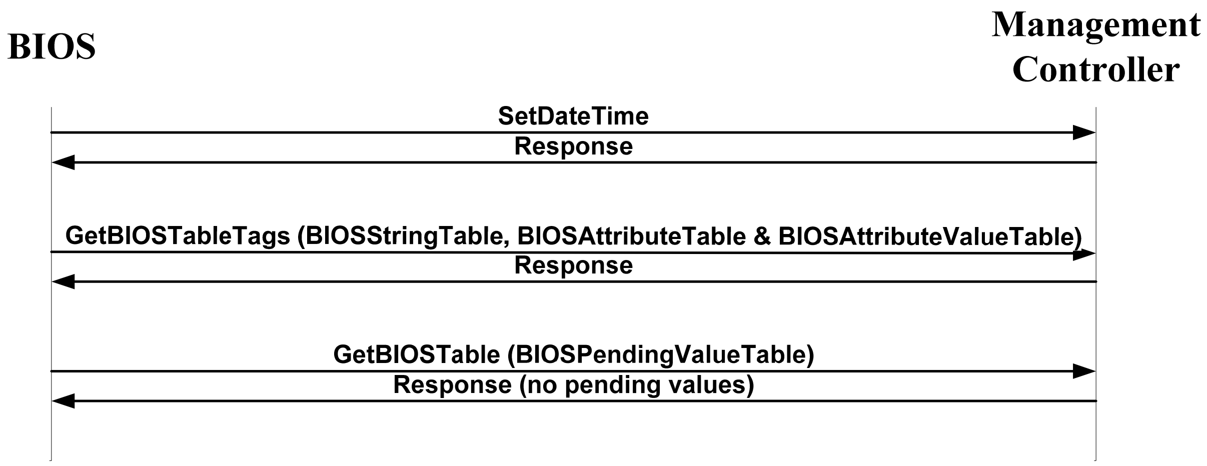

BIOS
Management Controller
SetDateTime
Response
GetBIOSTableTags (BIOSStringTable, BIOSAttributeTable & BIOSAttributeValueTable)
Response
GetBIOSTable (BIOSPendingValueTable)
Response (no pending values)

**Figure 4 – BIOS/MC Communications without MC Authentication for No BIOS Settings Changes**

Figure 5 is a variation of the preceding example with MC authentication.

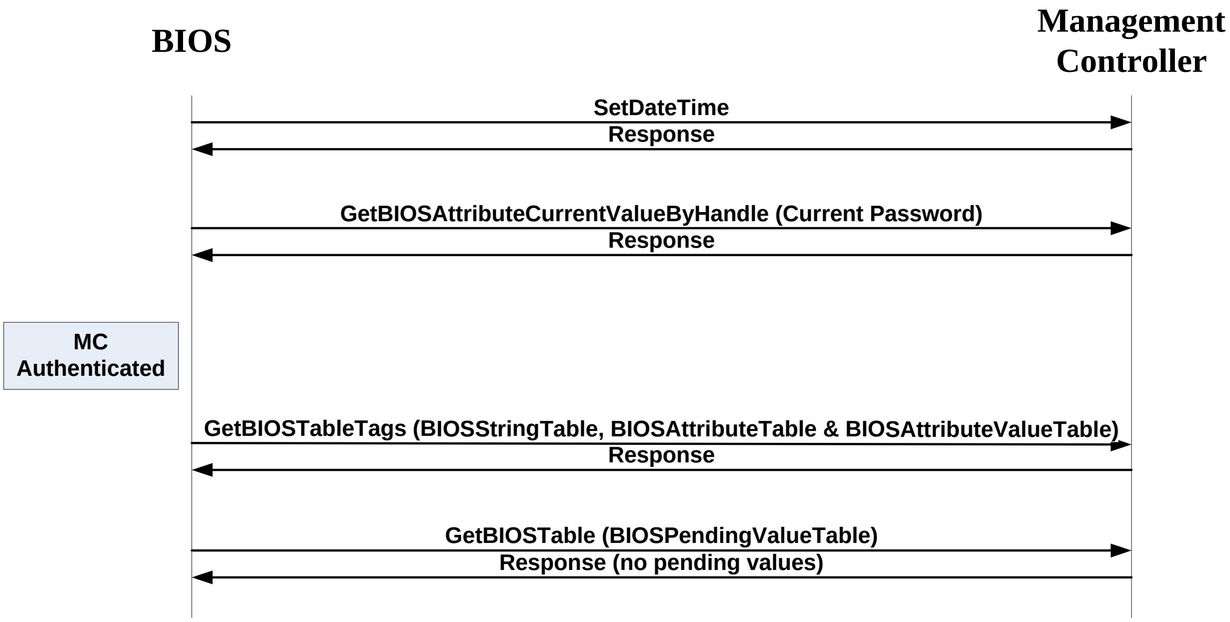

BIOS
Management Controller
SetDateTime
Response
GetBIOSAttributeCurrentValueByHandle (Current Password)
Response
MC Authenticated
GetBIOSTableTags (BIOSStringTable, BIOSAttributeTable & BIOSAttributeValueTable)
Response
GetBIOSTable (BIOSPendingValueTable)
Response (no pending values)

**Figure 5 – BIOS/MC Communications with MC Authentication for No BIOS Settings Changes**

## 9.4 Local BIOS Setting Changes

In Figure 6, the MC has the BIOS tables and the MC has no pending changes. The BIOS settings were locally modified and the BIOS provides the new BIOS settings to the MC. This example shows that the MC is authenticated before the transfer.

The BIOS first queries the current password from the MC by using the GetBIOSAttributeCurrentValueByType command. In the response, the BIOS receives the current password. The BIOS authenticates the MC with the password received in the response. After the MC is authenticated, the BIOS queries the BIOS table tags on the MC by using GetBIOSTableTags command.

The BIOS table tag for the BIOS Attribute Value Table in the response indicates to the BIOS that the BIOS settings on the MC are not the latest. The BIOS then uses the GetBIOSTable command to get any pending values of the BIOS attributes from the MC. The MC responds to the GetBIOSTable command with no pending values because no changes to the BIOS attributes were made remotely. Because the local BIOS setting were changed, the BIOS Attribute Value Table needs to be updated on the MC. The BIOS updates the copy of the BIOS Attribute Value Table on the MC by using the SetBIOSTable command.

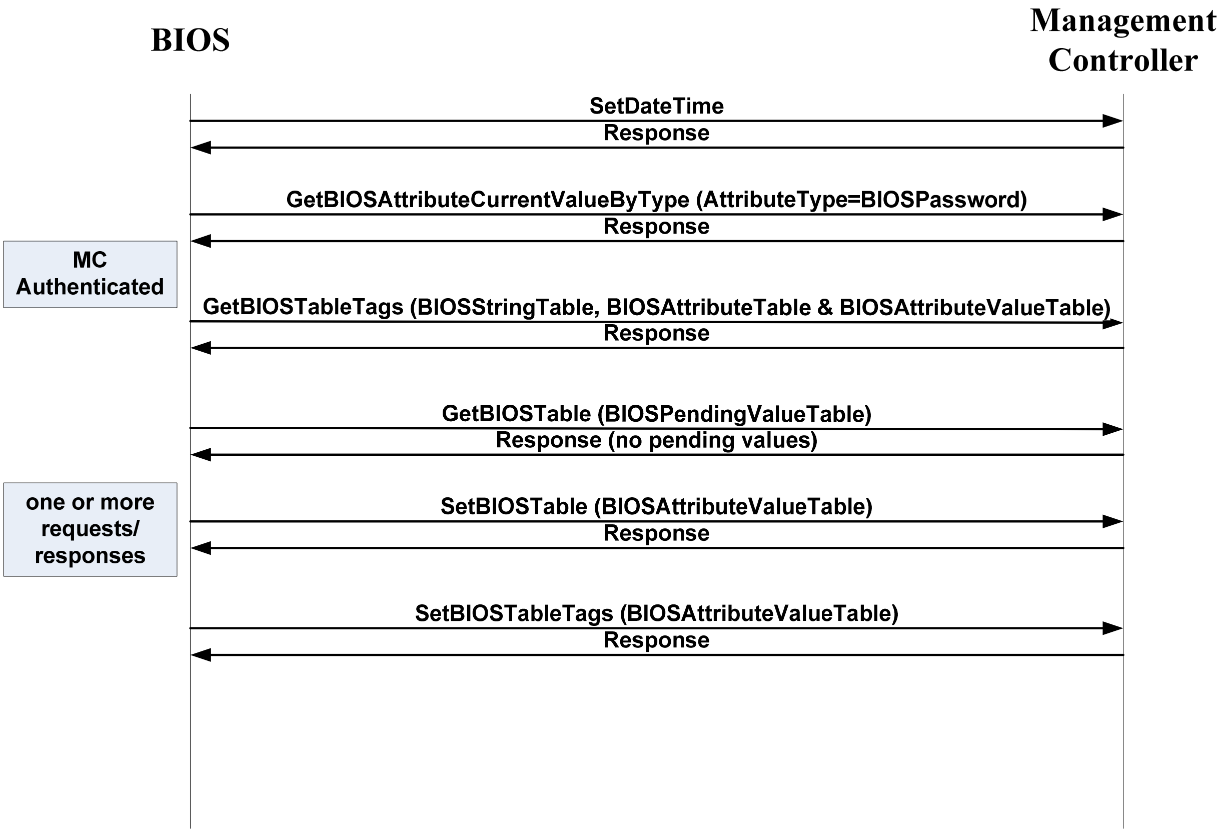

BIOS
Management Controller
SetDateTime
Response
GetBIOSAttributeCurrentValueByType (AttributeType=BIOSPassword)
Response
MC Authenticated
GetBIOSTableTags (BIOSStringTable, BIOSAttributeTable & BIOSAttributeValueTable)
Response
GetBIOSTable (BIOSPendingValueTable)
Response (no pending values)
one or more requests/ responses
SetBIOSTable (BIOSAttributeValueTable)
Response
SetBIOSTableTags (BIOSAttributeValueTable)
Response

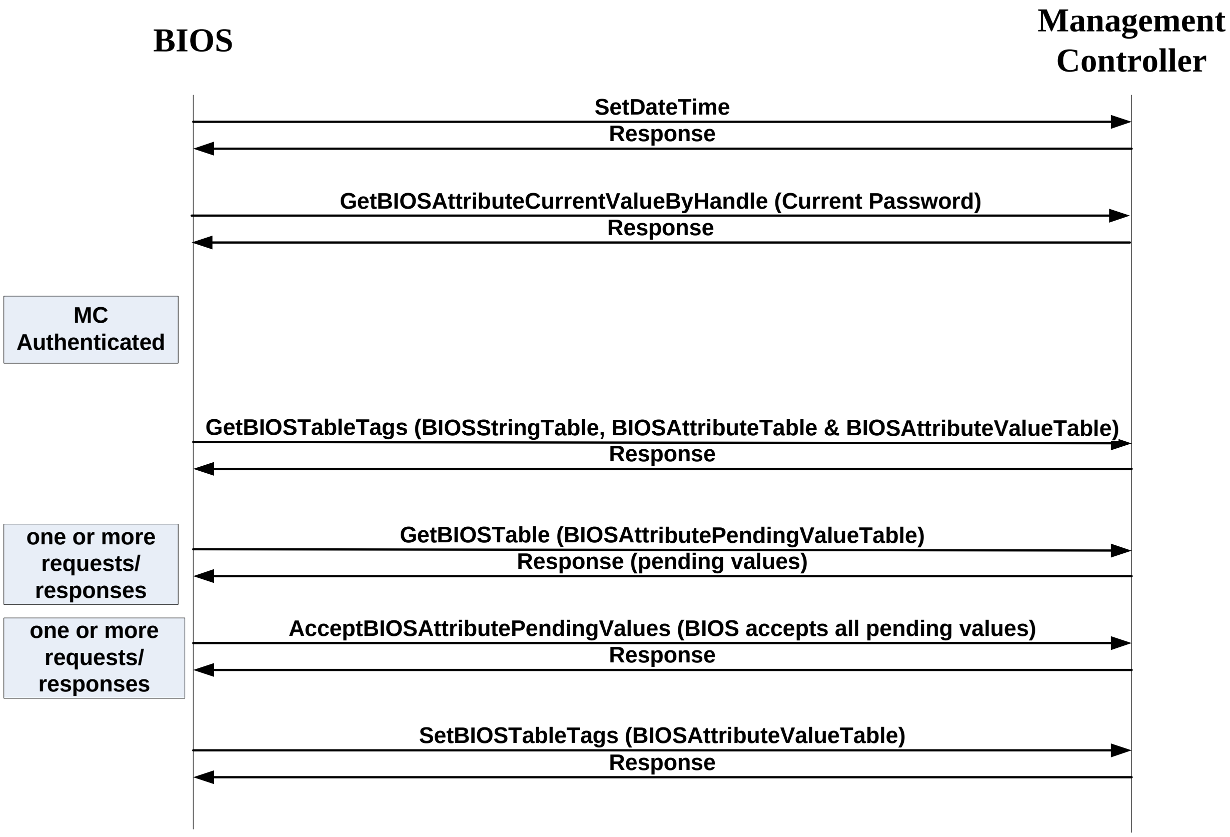
**Figure 6 – BIOS/MC Communications with MC Authentication for Local BIOS Settings Changes**

## 9.5 Remote BIOS Setting Changes Accepted

In Figure 7, the MC has the BIOS tables and the MC has pending changes. The BIOS settings were not locally modified and the BIOS accepts the pending values obtained from the MC. This example shows that the MC is authenticated before the transfer.

The BIOS first queries the current password from the MC by using the GetBIOSAttributeCurrentValueByHandle command. In the response, the BIOS receives the current password. The BIOS authenticates the MC with the password received in the response. After the MC is authenticated, the BIOS queries the BIOS table tags on the MC by using GetBIOSTableTags command. The BIOS table tag for the BIOS Attribute Value Table in the response indicates to the BIOS that the MC has the latest BIOS String Table, BIOS Attribute Table, and BIOS Attribute Value Table. The BIOS then uses the GetBIOSTable command to get any pending values of the BIOS attributes from the MC. The MC responds to the GetBIOSTable command with the pending values because some of the BIOS attributes were changed by the remote entity. The BIOS modifies the BIOS attributes that had pending changes. The BIOS notifies the acceptance of pending values to the MC by using the

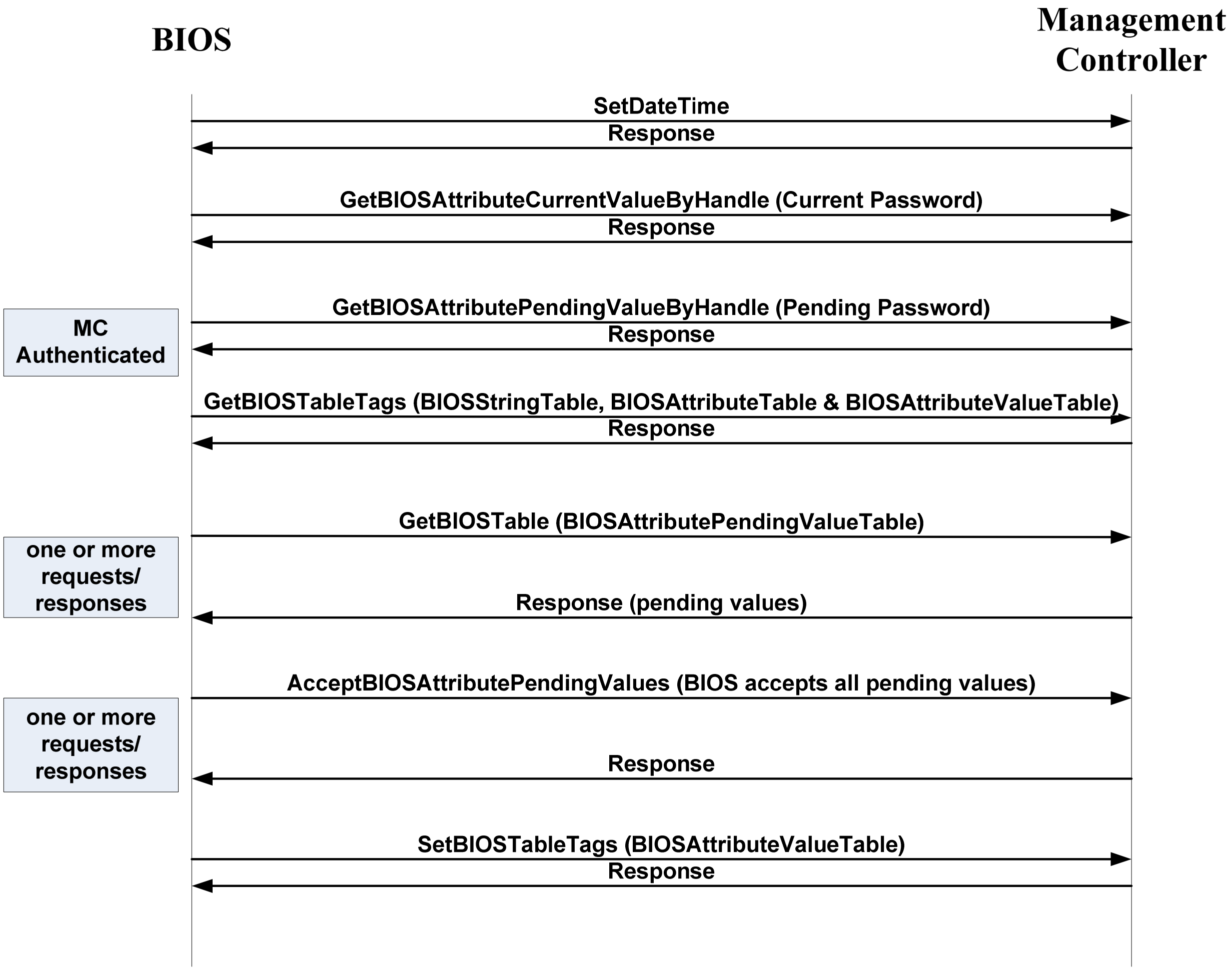
797 AcceptBIOSAttributesPendingValues command. This results in the MC updating its copy of the BIOS
798 Attribute Value Table to reflect the acceptance of the pending values. The BIOS updates the table tag of
799 BIOS Attribute Value Table to reflect the latest BIOS Attribute Value Table on the MC.


BIOS
Management Controller
SetDateTime
Response
GetBIOSAttributeCurrentValueByHandle (Current Password)
Response
MC Authenticated
GetBIOSTableTags (BIOSStringTable, BIOSAttributeTable & BIOSAttributeValueTable)
Response
one or more requests/ responses
GetBIOSTable (BIOSAttributePendingValueTable)
Response (pending values)
one or more requests/ responses
AcceptBIOSAttributePendingValues (BIOS accepts all pending values)
Response
SetBIOSTableTags (BIOSAttributeValueTable)
Response

800

801 **Figure 7 – BIOS/MC Communications with MC Authentication (Based on Current Password) for**
802 **Remote BIOS Settings Changes**

803 Figure 8 is a variation of Figure 7, with the use of a pending password for the MC authentication.


BIOS
Management Controller
SetDateTime
Response
GetBIOSAttributeCurrentValueByHandle (Current Password)
Response
MC Authenticated
GetBIOSAttributePendingValueByHandle (Pending Password)
Response
GetBIOSTableTags (BIOSStringTable, BIOSAttributeTable & BIOSAttributeValueTable)
Response
one or more requests/ responses
GetBIOSTable (BIOSAttributePendingValueTable)
Response (pending values)
one or more requests/ responses
AcceptBIOSAttributePendingValues (BIOS accepts all pending values)
Response
SetBIOSTableTags (BIOSAttributeValueTable)
Response

804

805 **Figure 8 – BIOS/MC Communications with MC Authentication (Based on Pending Password) for**
806 **Remote BIOS Settings Changes**

807

# ANNEX A
(informative)

## Change Log

| Version | Date | Author | Description |
|---|---|---|---|
| 1.0.0 | 2008/10/24 | Hemal Shah | Preliminary release draft |
| 1.0.0 | 2008/11/5 | Hemal Shah | Addressed Platform SC comments. Preliminary release. |
| 1.0.0 | 2009/4/23 |  | DMTF Standard Release |