MXM
GRAPHICS MODULE

Software Specification

Version 3.0

Revision 1.1

July 20, 2009

Document Change History

Revision	Date	Responsible	Description of Change
1.0	October 29, 2008	LPL, SM	Initial Release
1.01	December 19, 2008	LPL, SM	Overview: Specification 1.0.1 primarily corrects discrepancies between the released 1.0 SDK and the 1.0 specification. The 1.0 SDK already supports all items in spec 1.0.1 except added power state management options. No SIS or interface changes are required from SDK 1.0 to use existing 1.0 functionality. 2.5.1 Minor phrasing change due to addition of pre-P-state change callback (see 4.3.7.4) 2.5.2 Examples for use of slowdown input to module. 3.2.1 Typo and two outdated entries corrected in MXM3_EFI_INTERFACE struct. Rest of document was already correct in 1.0, struct now matches. 3.2.2 MxmReturnSpecLevel listed an EDID read function which did not exist in the 1.0 document. Removed, and renumbered the list entries to match. 3.3.1 Typo (MXM_FUNC_EVENLIST) corrected. Typo in sample GUID for event 0x80 corrected (0xFB, should have been 0xF8 – this was already correct in the associated comment, the MXM_FUNC_EVENTLIST table, and the sample MOF. Updated sample code and MOF to include events 0xEF, 0xF0, and to remove event 0xD0 (see 4.3.7.1 and 4.3.7.2). 3.3.2 & 3.3.2.1 Corrected values in sample code. 4.1.1 Changed BL value numbering on return when retrieving EDID to avoid conflict with MXM 2.x. Not used by SDK 1.0 VBIOS or tools. 4.1.4 Changed register to return pointer to VBIOS, to more easily support blocks greater than 64K. Not used by SDK 1.0 VBIOS or tools. 4.3.7.1 Replaced notification 0xD0 with 0xEF 4.3.7.2 Replaced notification 0xD0 with 0xF0. Clarified that SBIOS input on hotkey selection is optional. 4.3.7.4 Added pre-P-state-change callback option to MXM_FUNC_MXCB. Added bits to return parameters to match the input parameters. 4.3.7.5 Two 'future' section GUIDs updated (not used in existing spec). Replaced notify 0xD0 (notify for update of both MXPP and MXDP) with 0xEF and 0xF0 (notifies for updates of MXPP and MXDP respectively).

Revision	Date	Responsible	Description of Change
			Updated example to not reference 0xD0. 5.2 Clarified DDC association for DVI-I. Added separate enumerations for the DP Aux ports in I2C field (0x9-0xC instead of all using 0x8 – already in SDK 1.0). Clarified link association in Digital Connection field. 5.5 Clarified use of PWR_LEVEL#. Added backward-compatible bit for GPIO-only power state select.
1.1	July 20, 2009	LPL, SM	Overview: Specification 1.1 updates fields for new display standards, addresses possibly confusing references to uses of the PWR_LEVEL# pin, and reverts a DDC association made in release 1.0.1 which was not adopted in the MXM hardware specification. Existing implementations of the 1.0 and 1.0.1 specifications are not affected by these updates. 2.5 and 5.5 Dropped references to alternate uses of PWR_LEVEL# pin, this pin is intended solely to indicate whether full power is or is not available. 3.3.2 Corrected sample code references for ACPI version checking to compare to 0x300 instead of 0x30. 4.1.3 Clarified valid parameters for Boot Message System BIOS callback.4.3.7.4 Added new MXM_FUNC_MXCB callback options to support SBIOS notification on certain events. Since this is a new feature, it may not be supported by all GPU drivers. 4.3.10.3 Extended external digital connector table entries for HDMI 1.2 to also apply to HDMI 1.3. 4.3.10.3 Added DisplayPort 1.2 to list of types of external digital connectors. 4.3.10.4 Added eDP to list of types of internal flat panels. 5.2 Added eDP to the list of Connector Types. 5.2 Default DDC association for DVI-I when using TMDS-over-LVDS moved to LVDS_DDC instead of VGA_DDC. MXM software should use the DDC field in the Output Device Structure to correctly support both old and new default assignments.

Table of Contents

List of Figures

List of Tables

1 MXM Version 3.0 Software Overview

1.1. Software Control of the MXM

Key goals of the MXM v 3.0 software standard include interoperability of compliant MXM adapters and compliant MXM systems. An MXM adapter and its associated software (video BIOS and OS specific drivers) should work on any MXM compliant system without re-engineering of the module software or system software.

To achieve this, the MXM v 3.0 software standard defines a **System Information Data Structure** which resides on the system, and a standard set of software interfaces between the MXM adapter and MXM system (refer to the section *MXM v 3.0 Interface Requirements* to determine the minimum software requirements).

The MXM system information data structure defines the system configuration outside the MXM connector, including details such as:

- Display connectors
- Cooling capability
- System power supply capabilities

This MXM system information structure may be stored either in a separate I2C ROM on the motherboard, or in a data block within the system firmware (system BIOS, or SBIOS).

The MXM software interfaces provide methods to abstract control of functionality described in the system information structure. For example, display MUX steering and the consistent enumeration of display devices.

The module's VBIOS and driver software interpret the contents of the MXM v 3.0 structure and call the methods provided by the system to integrate the graphics module into the system. The SBIOS does not need to parse any data about the module, nor does it need to modify its behavior based on the particular module which is present. It simply stores the system information structure in binary form and reports it to the module when requested.

In this specification, the designer of the MXM module and its software are referred to as the "(graphics) vendor." The designer of the system and SBIOS is referred to as the "system designer."

Note:	The MXM v 3.0 standard provides the foundation for interchangeable compliant modules. The SBIOS must not use specific properties, such as a module's PCI device ID, to alter features or behaviors defined via the MXM standard.

1.1.1. MXM Advantage

The MXM software standard provides a substantial benefit in terms of flexibility, maintenance, and inventory control. Without standardized interfaces, a unique VBIOS containing system specification settings must be generated for every module and system combination (X*Y unique VBIOSes). The same may be true for the SBIOS.

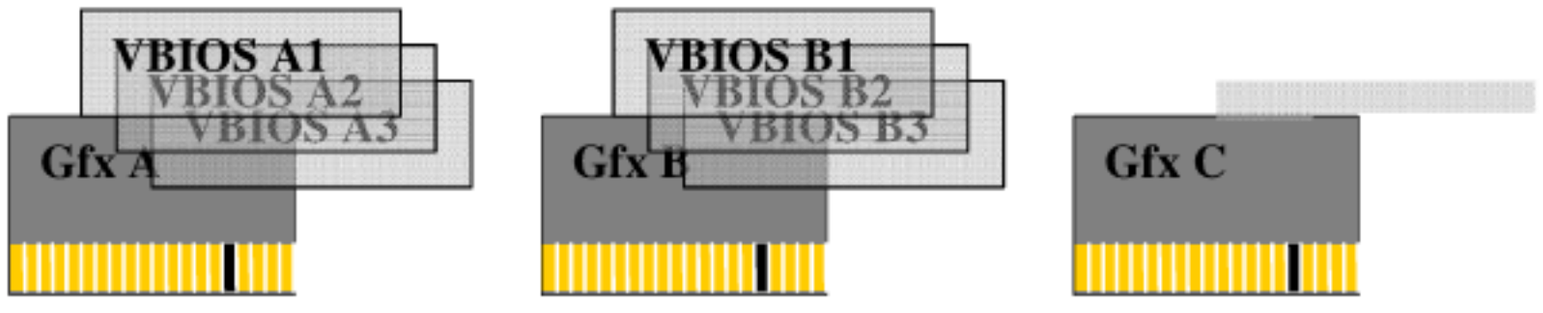

Figure 1-1. Non-MXM Software Combination Matrix Illustration

With an MXM compliant module and platform, a single VBIOS per module type and a single SBIOS (with system information) per system type is sufficient. The graphics software combines this information at runtime.

Figure 1-2. MXM Software Illustration

1.1.2. MXM Versioning and Interoperability

When an MXM adapter is installed into a system, the resulting system capabilities are the intersection set of the MXM adapter and the base platform. Only the capabilities that are present in both will be available. For example, if (A) the MXM module supports the High-Definition Multimedia Interface (HDMI™) output technology and (B) the base platform supports routing HDMI to an HDMI connector, then that likely means that HDMI can be supported. However, if the base platform (including dock) did not have an HDMI connector then no HDMI capability should be expected regardless of the MXM module's capabilities.

Figure 1-3. MXM Capabilities

Both the MXM adapter and the base platform convey software version information indicating the highest level of the MXM v 3.0 software interface that each supports. Between the MXM v 3.0 software interfaces, backwards compatibility within a major version is required. For example, a 3.x platform shall support any adapter 3.x ~ 3.0. Backwards compatibility with MXM v 2.x and 1.x is not required. Forward compatibility is not required. However, all structures and members, within later revisions (3.y) will remain backward compatible with previous 3.x implementations.

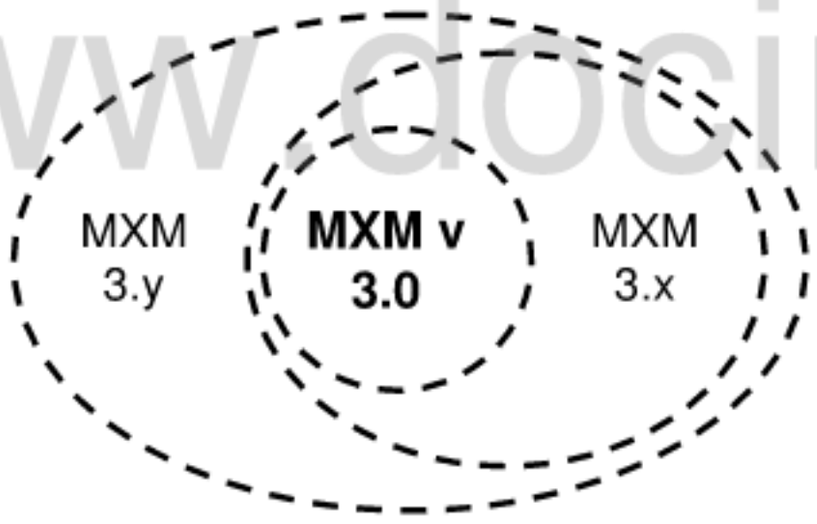

Figure 1-4. MXM v 3.0 Software Compatibility

1.1.3. MXM v 3.0 Interface Requirements

The following requirements are intentionally minimal. The system does not need to interpret anything about the module capabilities. The module software must have access to certain pieces of information and interfaces, both before driver load (typically pre-OS operation where the VBIOS is responsible for graphics functionality, where Int15h and EFI interfaces are used) and after the driver is present (where the ACPI interfaces are in effect). Functionality must also consider the needs of MXM adapters. They may need to operate as secondary display adapters (VGA resources are disabled, or it is enumerated as a non-VGA device) when the VBIOS interfaces may not be available. Both Int15h/EFI and ACPI interfaces are required. These are required even if the MXM module is not the primary VGA device. Refer to the following sections for descriptions on the Int5h/EFI and ACPI system methods.

- *MXM v 3.0 INT 15H System BIOS Callbacks*
- *MXM v 3.0 EFI System BIOS Callbacks*
- *MXM v 3.0 ACPI Methods*

MXM modules may contain either legacy (non-EFI) VBIOSes or hybrid EFI/legacy VBIOSes. If legacy operating system support is needed, the VBIOS and SBIOS must provide legacy interfaces.

The following summarizes the **required** software support:

- Setting Subsystem Vendor and Device ID. The SBIOS should set the module's subsystem information to the desired value. The specific registers to set may vary by graphics processing unit (GPU). Consult graphics vendor-specific documentation.
- MXM Structure for each MXM connector
  - Header and Checksum are required
  - At least one Output Device substructure (if the adapter has an output)
  - Cooling capability substructure
  - At least one Input Power substructure
- An interface to obtain the MXM structure from the system. Typically, this is a software interface. Less commonly, the MXM structure can be stored in a platform-resident ROM (only one of the two approaches is needed).
  - Int15h Func 0 / EFI MxmReturnSpecLevel and Int15h Func 1 / EFI MxmReturnStructure
  - ACPI_DSM method subfunctions MXM_FUNC_MXSS, MXM_FUNC_MXMI and MXM_FUNC_MXMS

Additional items **may be required** depending on the system features. Following examples are:

- Backlight substructure if GPU PWM backlight is used
- GPIO substructures if module GPIOs are used
- Int15h Func 2 / EFI standard EFI_EDID_OVERRIDE_PROTOCOL and the _DDC standard ACPI method if a non-DDC internal flat panel is used
- Int15h Func 3 / EFI MxmSelectOutputDevice and the MXMX & MXDS ACPI methods if a Display DDC or Output MUX is used with system methods.

Unless otherwise noted, structures and methods not listed are optional and not required.

The MXM structure is static. It is created once during platform design (with the help of the MXMedit tool) and does not change based on the current operating mode of the system. Parsing of the MXM structure is required only by the GPU software.

Since the GPU software must be active in order to request and parse MXM information, behavior requested by the system in the MXM system information structure will not take effect until the software has initialized. This includes thermal and power information. In addition, due to VBIOS limitations full functionality may not be available until the GPU software driver loads during OS initialization.

Compliant MXM adapters which, on boot, detect that an **MXM System Information Structure** is not present must assume they are operating on a non-compliant system. The module must ensure safe operation in a minimal configuration safe-mode. This includes booting on the CRT only, in the lowest possible performance state with lowest possible thermal and input power burden. The use of warning messages by the module, at POST or OS boot, to indicate mis-configuration, is strongly encouraged.

1.2. Software Support for OEM Modules

The MXM v 3.0 standard allows for OEMs to create limited variants outside the core MXM standard in order to add features or optimize cost. The platforms must also be capable of supporting a fully MXM compliant module.

Such deviations from the specification, such as additional power rails, require no software support. The use of a VBIOS image merged with the SBIOS on a single ROM on the platform does not change the behavior of the MXM module software.

1.3. MXM VBIOS

The VBIOS for an MXM compliant module should be stored in a ROM on the MXM v 3.0 graphics module (in the same manner as VBIOSes are stored for desktop cards). OEM modules can instead store the VBIOS in the SBIOS in a single non-volatile storage device on the motherboard.

Systems which are MXM compliant shall support MXM compliant modules. Any modules with a VBIOS in on-module ROM will override the VBIOS in the system. For modules without a ROM, during POST the VBIOS will check if it is running on the adapter it was built for (for example, by checking Device ID) and fail POST if it is not correctly matched.

The MXM VBIOS is fully VESA compliant and operates in the same fashion as non-MXM VBIOSes. Some vendor-specific extensions may also be available. However, use of these extensions should be avoided in order to facilitate cross-vendor module compatibility.

Refer to the *MXM v 3.0 Interface Requirements* section for interface and operational requirements.

2 The MXM System Information Structure

The MXM v 3.0 structure consists of a mandatory header structure followed by a variable number of substructures and a mandatory checksum byte. Data is assumed to be in little endian format.

Systems may have more than one MXM connector, so they may store more than one MXM structure. The software interface supports specifying which connector data is being requested for. Data in MXM structures is specific to a particular system configuration (for example, which connectors are present on a particular model of the system, which MXM slots are currently stuffed, etc.), so an SBIOS may store multiple version of the MXM structures and report them based on the particular model and configuration of the system.

The MXM structure for a system is created through a simple editing tool (available as part of the SDK). The MXM structure is stored in the SBIOS and is requested by and parsed by the module software. The exact bit-level contents of the structure are generally of interest only to graphics vendors and not system designers. Usage is summarized here and bit level details are provided at the end of this specification in Chapter 5.

2.1. MXM Header Structure

The MXM header identifies the version of the system information structure stored. Currently only one version is available (3.0), but in the future systems may, if desired, store MXM structures for multiple MXM module versions. If no exact match is available, the MXM software will determine the best mapping between itself and the available structure version.

Refer to Table 5-1 for a description of the header at the beginning of the MXM structure.

2.2. MXM Output Device Structure

Each output device structure describes a single display connector or integrated display. There will be multiple MXM output device structures in the MXM structure, defining one output device each.

An output device shall be enumerated for each supported integrated display and display connector. For example, each DVI output that can be routed to either an onboard connector or a docking connector must have one dedicated output device structure for each connector. In the case of a DVI-I output connector there are separate output device structures for the analog and digital outputs.

Any MXM module which outputs to a display device will therefore require one or more output device structures. Some MXM modules, such as secondary MXM modules in multi-GPU configurations, may have no output device structures.

The ordering of output devices implies the default boot device, and the detection order. For example, the first output device is the default boot display device. The second output device will be the boot display device if the first is not attached. The third if the second is not attached, and so on.

Refer to Table 5-2 for a description of the **MXM Output Device Structure**.

2.2.1. Sharing Display Output or DDC/ Aux Lines Across System Connectors

If a system requires more display outputs or DDC/Aux lines than the MXM adapter provides, it may need to use on-system MUXes to share signals. The display outputs and DDC/Aux lines must be controlled separatel. For example, a single MUX control cannot be used to switch both the output and the DDC/Aux lines. DDC/Aux lines also cannot be shared by attempting to put them all on a single bus without isolating them via a MUX (some external display devices can interfere with one another if connected on a shared bus, making it impossible to communicate properly unless they are isolated properly.).

Note: In the MXM specification, MUX control can be done either through direct GPIO control from the module (the signals to control the MUX are routed directly to the GPIO pins on the MXM connector, and defined in the GPIO structures) or through a defined interface to the SBIOS (the signals to control the MUX are handled by the platform in response to calls from the MXM graphics software to the SBIOS).

If signal control direct from the module is used:

- Display output MUX control must be implemented as a 2:1 switch. The use of this control must be indicated in the Output Device Structure, and the GPIO used must be described in the GPIO Device Structure. The associated MXM connector GPIO and the state of that GPIO in order to select the output must be provided.
- DDC/Aux line control must be implemented as a n:1 switch. Each connector will have an associated GPIO identified in the Output Device Structure and the GPIO Device Structure. When this GPIO is asserted, the GPU can communicate with the device.

2.3. MXM System Cooling Capability Structure

The **MXM System Cooling Capability Structure** defines the thermal power dissipation capability of the MXM thermal solution contained in the system. This structure is **required** for all MXM systems. The maximum performance level of the MXM module will be determined by the lowest of its cooling capability and its current input power.

Refer to Table 5-3 for a description of the MXM Cooling Capability Structure.

2.4. MXM Thermal Structure

The MXM v 3.0 module hardware and software will regulate its own temperature by varying the performance of the GPU. The **optional** thermal structures allow the system to specify its own thermal requirements. There can be up to one thermal structure of each type as described.

- **Maximum temperature**: If the system requires the module maintain a lower maximum temperature than the module default, an MXM thermal structure can specify this information in terms of GPU junction temperature. The MXM module hardware and software will ensure the module meets the system's stated requirements.
- **Alert signals**: The TH_ALERT pin output from the module indicates a high (but not critical) temperature, which may for example be used by the system as a signal to enable a higher system fan speed. The system can specify the temperature at which this pin should assert.

Note: The TH_OVERT pin is a separate connector pin which indicates a critically high module temperature (at which shutdown of the module is recommended). This temperature is determined by the module and therefore cannot be specified by the system. If this temperature is exceeded at any time, including power-on, the system must shut down the module immediately to prevent damage.

Refer to Table 5-4 for a description of the MXM Thermal Structure.

2.5. MXM Input Power Structure

The **MXM Input Power Structure** defines the maximum continuous available input power provided by the system for the PWR_SRC input power rail. At least one power structure is **required**. An additional MXM input power structure is required for each platform supported power level for the MXM module. Typically there will be either one structure (which then applies globally) or two (one for AC and one for battery operation).

The maximum performance level available to the MXM module will be determined by the lowest of its cooling capability and its current input power. MXM modules generally contain a limited number of pretested operating power states, from which the module software selects an appropriate level that

is equal to or less than the system's current power budget. Due to the limited number of module levels, small changes to the MXM structure information may cause substantial changes in available performance. For example, if a module contains 30 W, 35 W and 40 W performance levels, a 35 W system power will result in a 35 W module power state. Changing the system power to 36 W will have no change in graphics performance, while changing it to 34 W would require the module to reduce its power consumption to 30 W.

When first powered on, until it can read the MXM system information, the module should operate in a low power state. At software initialization, GPU software can read the module and system capabilities. The GPU software initialization may fail if the lowest module P-state exceeds the current system power budget for the module.

Refer to Table 5-5 for a description of the MXM Input Power Structure.

2.5.1. Auxiliary Power States

Some systems may require more than the two standard power states (AC and battery). For example, a system may support a light-weight power-brick that is AC connected but cannot support operation at the same full performance as the normal AC state. Or it may possess a higher capability docking station that allows operation at higher states than normal AC and battery operation.

If auxiliary power states are defined, they behave like static performance limits such as those defined for the PWR_LEVEL# signal. They define additional states wherein the module performance is limited by the MXM software (through the use of voltage, frequency or other limiting mechanisms) to ensure the maximum power consumed by the module always remains within the specified limit.

Entry into a particular power level is triggered via corresponding **ACPI Notify Code**. Notification codes 0xD1~0xD5 represent P0~P4 respectively. P0 is the default operating mode, representing no auxiliary power state restriction.

Auxiliary power states are intended for system state changes such as the power supply changes as previously described. Use of such states requires a GPU driver to be active. Fine-grained runtime power control should not be assumed by the system through this mechanism.

Constraint to a new power limit does not occur immediately when the notifier is issued. The transition is not required to complete within any consistent time interval, and may vary depending on system demands or quality of service constraints. This mechanism should therefore not be used for critical power control (for example, for thermal control). The module can provide an optional callback at the start and end of the power state transition if the system needs to know when the new power limit has been met (this usage is outlined in the section *Changing Power States*.).

These limits do not imply operation at a specific power or clock level -- the module may operate at any performance level which meets the specified power limit.

2.5.2. Changing Power States

Many systems can provide the same level of power to the graphics module regardless of operating conditions. In this case, no action is needed. However, for designs where control of module power is required (such as notebooks which can supply less power to the module during battery operation), several options are available.

Power restrictions from the system may occur due to tuning preferences by the system or due to external events such as the system being disconnected from AC power.

If a system supports only one module power state, only one input power structure will exist (with the **Type** set to 1).

See Section 5.5 for the bit field definitions of the Input Power Structure.

2.5.2.1. Asserting the Power State Pin (PWR_LEVEL#)

Asserting PWR_LEVEL# is required in all cases when a power state transition is due to a drop in the available power to the module. This provides the opportunity for the module to respond immediately, without waiting on software notifications. Asserting is not required for power state transitions caused by an increase in available power or a decrease requested for reasons other than power availability (such as entering a power saving operating mode).

Note: If the PWR_LEVEL# input pin is asserted on the module, then regardless of other requirements or SBIOS power state requests, the module must restrict power consumption to power specified in the Input Power Structure with Type 0. As described, this hardware induced power consumption change may be used as an interim 'safe' state during power level transitions, or as an end state.

If the PWR_LEVEL# pin will stay asserted for the duration while at input power level Type 0, that input power structure must set the notification type bit to 1. Otherwise it is expected that the pin will only be asserted during the transition (as outlined in the examples in Section 2.5.2.3).

If the system has *no* ACPI mechanisms implemented, at most two power levels can be supported (Type 0 and Type 1). Power selection must be done entirely off the PWR_LEVEL# pin. The ACPI type field should be set to 1 to indicate this system method.

2.5.2.2. ACPI Notifications

If input power levels 0 and 1 are defined, the system is assumed to support the standard ACPI notebook power state notification (with level 0 indicating the battery power level).

If any auxiliary power states are defined, the system is assumed to support **MXM ACPI Auxiliary Power State** notifications. The MXM ACPI notifiers (one or more of 0xD1-0xD5) must be implemented.

These two notification methods set separate thresholds. The MXM driver must select the lower of the two if both are in effect. For example, if a system is running on battery power and the SBIOS also requests the module limit power to Auxiliary Power Level 1 (Input Power Structure Type 0x9), the lower of the two limits applies.

2.5.2.3. Examples

The interaction of these mechanisms offers a range of options to the system, as illustrated.

- **Example 1: Basic power supply transitions (AC and battery) only**
  - Setup requirements:
    a) PWR_LEVEL# should be connected to power supply or similar output to assert while running on battery.
    b) MXM input power structure: battery level power entry (entry with Type 0) should set the Hardware Notification field to '1'.
    c) No callbacks are required.
  - Runtime operation: on switching to battery power:
    a) System asserts PWR_LEVEL# pin on transition to battery power.
    b) Adapter restricts power consumption to Input power structure of Type 0 (battery level).
    c) SBIOS issues standard ACPI power source notification (battery mode)
    d) GPU driver receives request and may perform additional module power state adjustments. Power consumption remains restricted to the battery power level.
  - Runtime operation: on switching to full power:
    a) System de-asserts PWR_LEVEL# pin on transition to AC power.
    b) MXM adapter is free to resume operation at the AC power level.
    c) SBIOS issues standard ACPI power source notification (AC mode)
    d) Driver receives request and may perform additional module power state adjustments.

- **Example 2: Additional software controlled power states**
  Some systems may use additional power states selected by the SBIOS using the MXM auxiliary power notifications.
  - Setup requirements:
    a) MXM input power structure: power state Type 0 (only) *may* set the Notification Type field to '1' (as appropriate for system behavior).
    b) PWR_LEVEL# must be connected to a power supply signal or under software control (as explained).
    c) If the system supports an auxiliary power state or PWR_LEVEL# will not stay asserted while in power state Type 0, the SBIOS must support the MXM_FUNC_MXCB post-power state change callback (and should indicate this during the initialization time MXM_FUNC_MXCB support check performed by the MXM driver).

- Runtime operation:
  a) System asserts PWR_LEVEL# pin if required (see Section 2.5.2.1)
  b) If PWR_LEVEL# asserts, adapter restricts power consumption to input power structure of Type 0.
  c) SBIOS issues ACPI notification (standard ACPI power source notification, or MXM auxiliary power state notification) for desired state.
  d) Driver receives request and performs any needed module power state changes. The new limit is the min based on the current power source and the current auxiliary power state. On completion, driver issues MXM_FUNC_MXCB post-power state change callback.
  e) If PWR_LEVEL# was asserted, and will be de-asserted for this power state, SBIOS must wait till MXM_FUNC_MXCB callback before de-asserting.

- **Example 3: Systems requiring per-module power state control**

If a system wishes to control the power level of multiple MXM modules, it can use the same mechanism as previously described as long as the same power state will be set across all modules. However, if the system wishes to set different modules to different power levels, the SBIOS must support the pre-power state change MXM_FUNC_MXCB callback. This is to ensure proper communication of desired power levels even in cases such as WMI notifiers (which, unlike direct notifiers, are not module specific). If this callback is not supported, it is assumed by the module software that the same power state is desired for all modules.

- Setup requirements:
  a) MXM input power structure: power state Type 0 (only) *may* set the Notification Type field to '1' (as appropriate for system behavior).
  b) PWR_LEVEL# must be connected to a power supply signal or under software control (as futher explained).
  c) If the system supports an auxiliary power state or PWR_LEVEL# will not stay asserted while in power state Type 0, the SBIOS must support the MXM_FUNC_MXCB post-power state change callback (and should indicate this during the initialization time MXM_FUNC_MXCB support check performed by the MXM driver).
  d) SBIOS should support the MXM_FUNC_MXCB pre-power state change callback (and should indicate this during the initialization time MXM_FUNC_MXCB check).
- Runtime requirements:
  a) System asserts PWR_LEVEL# pin on all modules if required (see Section 2.5.2.1).
  b) If PWR_LEVEL# asserts, each adapter restricts power consumption to input power structure of Type 0.
  c) SBIOS issues ACPI notification (standard ACPI power source notification, or MXM auxiliary power state notification) for desired state.

d) Driver receives request and may call SBIOS via MXM_FUNC_MXCB pre power state change callback to verify desired power level on each GPU.

e) Driver receives request and performs any needed module power state changes. The new limit is the min based on the current power source and the current auxiliary power state. On completion, driver issues MXM_FUNC_MXCB post-power state change callback.

f) If PWR_LEVEL# was asserted, and will be de-asserted for this power state an SBIOS must wait until MXM_FUNC_MXCB callback before de-asserting.

Refer to the section on the MXM_FUNC_MXCB callback function for details on the MXM notification.

Note:	With these mechanisms the system is imposing a power limit on the module. This may not correspond to a change to a specific GPU clock level since power may be limited in other ways. Or since the module is free to select power levels at or below the current limit at any time, it may already be operating at a sufficiently low power level so no clock changes or other changes will be visible. The MXM_FUNC_MXCB callback calls must still be performed by the module software in order to confirm that sufficient action has been taken. Power state transitions may occur for reasons other than the current system request. A system waiting for a requested transition to complete should not assume that the next callback received is a response to its requested action. It should specifically check that the specified P-state is at or below the system requested level (before proceeding, for example, with de-asserting the PWR_LEVEL# pin).

2.6. MXM GPIO Device Structure and GPIO Pin Entry Structure

This structure applies to standard module-provided GPIOs, provided on the MXM connector. The structure is optional, and it is only needed if the system uses one or more MXM module GPIOs. GPIOs are described with a list of **GPIO Device Structures** and **GPIO Pin Entry Structures**. Each GPIO device description is followed by one or more GPIO pin description for pins on that device.

An MXM GPIO Device Structure is used to define whether the MXM Physical & Logical GPIO pins are attached to the pre-defined module GPIOs, or to OEM defined GPIOs.

The MXM GPIO Pin Entry Structures follow the GPIO Device Structure and enumerate the function and usage of all the MXM GPIO pins used.

Refer to Table 5-6 a description of the MXM GPIO Device Structure and Table 5-7 for the MXM Pin Entry Structure.

The Logical pin number referenced in other data structures such as the Output Device Structure is determined by numbering the GPIOs beginning with '0' for the first direct GPIO pin on the module connector up to the total number of direct GPIOs.

Refer to the *MXM Version 3.0 Graphics Module Thermal Electromechanical Specification* for the GPIO function hardware definitions.

2.7. MXM Vendor Specific Structure

This is an optional GPU vendor specific structure (VSS). The contents of the structure are defined by the vendor, where each structure is tagged to indicate to which vendor it applies. There may be multiple structures, for example, with one VSS per supported vendor. Contact the GPU vendor for any additional details on VSS requirements or contents.

Refer to Table 5-9 for a description of the general format of the **MXM Vendor Specific Structure**. Specific details of any VSS data will be provided separately in vendor specific documentation.

2.8. MXM Backlight Control Structure

This is an optional field that describes the settings for the LCD panel integrated into the chassis. If no PWM or I2C based LCD inverter is supported then the field is not required.

The table provides only the hardware information the GPU software requires in order to set a requested brightness. The interface by which brightness requests are communicated (for example, for brightness control hotkeys) is a separate topic outside the scope of the MXM standard. The Windows Vista operating system, for example, supports ACPI 3.0 standard ACPI notifiers and methods for this purpose. The OS provided control software in turn calls the GPU software to set the requested brightness value.

Refer to Table 5-10 for a description of the MXM Backlight Control Structure.

2.9. MXM Fan Control Structure and Fan Speed Structure

This optional structure applies to an MXM module-controlled system fan, if present. The MXM module provides a PWM output for this purpose.

Fan control is described with a Fan Control Structure and one or more Fan Speed Structures. The Fan Control Structure describes the physical properties of the fan. The Fan Speed Structures specify temperature thresholds and fan speeds to describe the expected behavior the module should implement.

Refer to Table 5-11 for a description of the MXM Fan Control Structure and Table 5-12 for the MXM Fan Speed Structure.

2.10.MXM Checksum Byte

The MXM checksum byte is the two's complement of the 8-bit sum of the entire MXM v 3.0 structure (including header but not the checksum itself) and is the last byte in the MXM table.

3 Core MXM System Interfaces

All MXM systems must provide access to the MXM System Information Structure. This can be done either via software interfaces or through a ROM on the motherboard that is directly accessible by the MXM module.

In most cases the same software functionality is mirrored in an Int15h, EFI, and ACPI version. The ACPI version is required, as the Int15h version. The EFI version is required for EFI compliant SBIOSes only.

Chapter 4 discusses additional system interfaces which may be needed depending on system design choices.

3.1. Required MXM Int15h System BIOS Callbacks

A set of SBIOS callback functions has been defined in order to allow the communication of system information between the VBIOS and the SBIOS.

If the SBIOS does not support a described callback, then when called it should return with a value other than 005Fh in AX.

Note: In some instances below, one of the parameters is the 'Adapter ID', used to differentiate which MXM module is being referenced. This is a 16-bit value constructed as follows from the PCI bus information:

Bits [15:8] Bus
Bits [7:3] Device
Bits [2:0] Function

An adapter ID of '0' is acceptable when referring to the primary GPU in the system (that is, the one whose VBIOS is POSTed at system boot).
The ability to access secondary adapter information without a driver is primarily needed to enable test applications.

3.1.1. Function 0 – Return Specification Support Level

This is a required function that allows the VBIOS to get information from the SBIOS about the level of the MXM software specification that the system supports, and the support information for individual functions.

Entry:

AX = 5F80h

BL = 00h

BH = FFh

EBX[16:31] = Adapter ID (see above)

CX = Revision of the MXM software specification that is supported by the MXM module

Format is binary coded decimal, for example: `0030h = 3.0, etc.`

Return:

AX = 005Fh to indicate that the system bios supports this function

BL = Revision of the MXM software specification that is supported by the system

Format is binary coded decimal, for example: `30h = 3.0, etc.`

CX = MXM functions supported

Bit 0 = 1

Bit 1 = 1 if Function 1 is supported, 0 if not supported

Bit 2 = 1 if Function 2 is supported, 0 if not supported

Bit 3 = 1 if Function 3 is supported, 0 if not supported

Bit 4 = 1 if Function 4 is supported, 0 if not supported

Bit 7 = 1 if Function 7 is supported, 0 if not supported

Bit 8 = 1 if Function 8 is supported, 0 if not supported

Bit 9 = 1 if Function 9 is supported, 0 if not supported

3.1.2. Function 1 – Return a Pointer to the MXM Structure

This is a required function that will return a pointer to the MXM structure, which is stored in the SBIOS ROM area or some other memory location which is accessible in real mode during video POST.

Entry:

AX = 5F80h

BL = 01h

BH = FFh

EBX[16:31] = Adapter ID (see above)

CX = Identifier for the data block to return

To obtain the MXM information structure, use CL to specify the revision of the MXM software specification that is supported by the MXM module. Format is binary coded decimal, for example: `0030h = 3.0, etc.`

To obtain an additional vendor specific data block, use CL to specify an identifier in the range 0x80-0x8F. If the system BIOS does not contain such a data block, it should return an error in AX. Graphics vendors should not assume a specific assignment within this range – the SBIOS can store the data for different vendors in any order in the 0x80-0x8F identifier range.

Details for any such OEM or graphics vendor specific data blocks are outside the scope of this document.

Return:

AX = 005Fh to indicate that the system bios supports this function

BX = Vendor ID of data block if CX = 0x80-0x8F, else 0

ES:DI = Pointer to the MXM structure in real mode memory (< 1MB)

3.2. Required MXM EFI System BIOS Callbacks

A set of SBIOS callback functions has been defined in order to allow the communication of system information between the VBIOS and the systems supporting the Extensible Firmware Interface (EFI).

Refer to Section 1.1.3 (*MXM v 3.0 Interface Requirements*) for a summary of interface and operational requirements.

3.2.1. EFI Interface

The GUID defined for the EFI interface.

```
#define MXM3_EFI_GUID {4EA9D4FE-E6F6-410b-8037-0F98B5968B65}
```

The interface itself is as follows:

```
typedef struct _MXM3_EFI_INTERFACE {
  MXM_RETURN_SPEC_LEVEL     MxmReturnSpecLevel;
  MXM_RETURN_STRUCTURE      MxmReturnStructure;
  MXM_SELECT_OUTPUT_DEVICE  MxmSelectOutputDevice;
  MXM_CHECK_OUTPUT_DEVICE    MxmCheckOutputDevice;
} MXM3_EFI_PROTOCOL;
```

Note: ControllerHandle and ChildHandle

In some instances below, one of the parameters is the handle 'ControllerHandle', used to differentiate which MXM module is being referenced. This is the EFI handle for the PCI device. 'ChildHandle,' where used, is the output device (the display).

usStructSize

usStructSize is set to the size of the supplied buffer by the caller, and on return should contain the actual size of the structure. If the structure is larger than the buffer, EFI_BUFFER_TOO_SMALL should be returned as the status of the call.

Any unsupported function should return a status of EFI_UNSUPPORTED. Bad input parameters such as an invalid handle should return EFI_INVALID_PARAMETER. Passing too small a buffer should return EFI_BUFFER_TOO_SMALL. A successful call should return EFI_SUCCESS.

3.2.2. MxmReturnSpecLevel – Return Specification Support Level

Caller provides pointer to location, which on call contains the module requested specification revision level, and which on exit, will contain the platform's preferred revision level and bit field of supported functions.

```
typedef EFI_STATUS (EFIAPI *MXM_RETURN_SPEC_LEVEL) (
   IN struct _MXM3_EFI_INTERFACE  *This,
   IN    EFI_HANDLE  ControllerHandle,
   INOUT UCHAR16  usStructSize,
   INOUT UCHAR8   *pucRevisionLevel,
   OUT   UINTN    *puSupportFuncs
);
```

Where the integer value passed in uSupportFuncs indicates supported functions:

Bit 0 = '1' (this method is required)

Bit 1 = '1' (MxmReturnStucture is also required)

Bit 2 = '1' if MxmSelectOutputDevice is supported, '0' if not

Bit 3 = '1' if MxmCheckOutputDevice is supported, '0' if not

Other parameters follow the definitions in the Int15h interface, function 0.

3.2.3. MxmReturnStructure – Return Pointer to MXM Structure

Caller provides expected MXM interface revision level and a pointer which, on return will contain the MXM structure.

```
typedef EFI_STATUS (EFIAPI *MXM_RETURN_STRUCTURE)(
   IN struct _MXM3_EFI_INTERFACE  *This,
   IN    EFI_HANDLE  ControllerHandle,
   INOUT UCHAR16  usStructSize,
   IN    UCHAR16  usDataBlockID,
   OUT   CHAR8    *pMxmStruct
);
```

Input parameters follow the definitions in the Int15h interface, function 1.

3.3. Required MXM ACPI Methods

Where supported, methods within the ACPI namespace of the graphics adapter provide access to platform specific MXM functionality known by the SBIOS. Refer to implementation specific documentation on ACPI video extensions (*Advanced Configuration and Power Interface Specification* Revision 3.0a) for additional details.

Refer to the *MXM v 3.0 Interface Requirements* for interface and operational requirements.

Figure 3-1 is an overview of the altered and new methods in the namespace (affected methods are **bold** type). (This includes optional methods described in Chapter 4.)

```
SB
|- PCI
    |- WMI1              // WMI Device
        |- WMMX          // WMI Method wrapper
    |- VGA               // Define the VGA controller in the namespace
        |- _DOS          // Method to control display output switching
        |- _DOD          // Method to retrieve info on child output devices
        |- _ROM          // Method to retrieve the ROM image for this device
        |- _DSM          // Method for probing MXM Support and calling MXM
        |- CRT           // Child device CRT
            |- _ADR      // Hardware ID for this device
            |- _DCS      // Get current hardware status
            |- _DGS      // Query desired hardware active \ inactive state
            |- _DSS      // Set hardware active \ inactive state
            |- MXMX      // Method for selecting display data channel
            |- MXDS      // Method for selecting display output
        |- HDMI          // Child device HDMI
            |- _ADR      // Hardware ID for this device
            |- _DCS      // Get current hardware status
            |- _DGS      // Query desired hardware active \ inactive state
            |- _DSS      // Set hardware active \ inactive state
            |- MXMX      // Method for selecting display data channel
            |- MXDS      // Method for selecting display output
        |- LCD           // Child device LCD
            |- _ADR      // Hardware ID for this device
            |- _DDC      // Get EDID information from the monitor device
            |- _DCS      // Get current hardware status
            |- _DGS      // Query desired hardware active \ inactive state
            |- _DSS      // Set hardware active \ inactive state
            |- _BCL      // Brightness control levels
            |- _BCM      // Brightness control method
        |- TV            // Child Device TV
            |- _ADR      // Hardware ID for this device
            |- _DDC      // Get EDID information from the monitor device
            |- _DCS      // Get current hardware status
            |- _DGS      // Query desired hardware active \ inactive state
            |- _DSS      // Set hardware active \ inactive state
```


Figure 3-1. Namespace with MXM Methods Example

3.3.1. Accessing MXM ACPI Methods via WMI

In order to allow access to the new MXM data from user mode and kernel software across Windows*NT O/Ses the following headers shall be included in the graphics adapter namespace and any other devices that contain MXM methods in order to facilitate access via WMI. The GUID for WMI MXMX methods is:

{F6CB5C3C-9CAE-4EBD-B577-931EA32A2CC0}

The WMI GUIDs to use for **Notify** events are the same as those listed under Section 4.3.7.5 (MXM_FUNC_EVENTLIST).

WMI parameter Arg1 is used to specify which GPU's method is being called. The value of Arg1 is determined by calculating (0x100 + PCIe bus number for the GPU). Any related methods calling the motherboard chipset's integrated graphics should use Arg1 = 0x10.

For more details on using WMI to access ACPI methods refer to: http://www.microsoft.com/whdc/system/pnppwr/wmi/wmi-acpi.mspx.

In addition to the MXM-specific items and events, several events which are typically OS-supported but are key to MXM operation are included in the following ASL code and in the compiled MOF. If an OS does not support some of these events, they may need to be included under WMI1 in order to enable MXM software. Including them here also enables application level testing and verification. The GUIDs for these other events are also available under the section on MXM_FUNC_EVENTLIST.

Examples for this code are also provided in the SDK.

```
Device(WMI1)      // placed within PCI Bus scope
{
    Name(_HID, "pnp0c14") // pnp0c14 is the ID assigned to WMI mapper
    Name(_UID, "MXM2")              // use a unique UID for each instance

    // Description of data and events supported
    Name(_WDG, Buffer() {
        // Methods GUID {F6CB5C3C-9CAE-4ebd-B577-931EA32A2CC0}
        0x3C, 0x5C, 0xCB, 0xF6, 0xAE, 0x9C, 0xbd, 0x4e, 0xB5, 0x77, 0x93, 0x1E,
        0xA3, 0x2A, 0x2C, 0xC0,
        0x4D, 0x58, // Object ID "MX" - method "WMMX"
        1,          // Instance Count
        0x02,       // Flags (WMIACPI_REGFLAG_METHOD)

        // MOF data {05901221-D566-11d1-B2F0-00A0C9062910}
        0x21, 0x12, 0x90, 0x05, 0x66, 0xd5, 0xd1, 0x11, 0xb2, 0xf0,
        0x00, 0xa0, 0xc9, 0x06, 0x29, 0x10,
        0x58, 0x4D, // Object ID "XM"
        1,          // Instance Count = 1
        0x00,       // Flags

        // DISPLAY_HOTKEY, ACPI_NOTIFY_PANEL_SWITCH GUID
        // Notify event 80 (fixed) - Hot-Key, use _DGS, _DCS … etc.
        // GUID {E06BDE62-EE75-48F4-A583-B23E69ABF891}
        0x62, 0xDE, 0x6B, 0xE0, 0x75, 0xEE, 0xF4, 0x48, 0xA5, 0x83, 0xB2, 0x3E, 0x69, 0xAB,
0xF8, 0x91,
        0x80, 0x00, 0x01, 0x08,

        // DISPLAY_HOTplug, ACPI_NOTIFY_DEVICE_HOTPLUG
        // Notify event 81 (fixed) - Hot-Plug, query _DCS
        // GUID {3ADEBD0F-0C5F-46ED-AB-2E-04-96-2B-4F-DC-BC}
        0x0F, 0xBD, 0xDe, 0x3A, 0x5F, 0x0C, 0xED, 0x46, 0xAB, 0x2E, 0x04, 0x96, 0x2B, 0x4F,
0xDC, 0xBC,
```

```
        0x81, 0x00, 0x01, 0x08,

        // BRIGHTNESS_INC, ACPI_NOTIFY_INC_BRIGHTNESS_HOTKEY
        // Notify event 86 (fixed) - Backlight Increase
        // GUID {1E519311-3E75-4208-B05E-EBE17E3FF41F}
        0x11, 0x93, 0x51, 0x1E, 0x75, 0x3E, 0x08, 0x42, 0xB0, 0x5E, 0xEB, 0xE1, 0x7E, 0x3F,
0xF4, 0x1F,
        0x86, 0x00, 0x01, 0x08,

        // BRIGHTNESS_DEC, ACPI_NOTIFY_DEC_BRIGHTNESS_HOTKEY
        // Notify event 87 (fixed) - Backlight Decrease
        // GUID {37F85341-4418-4F24-8533-38FFC7295542}
        0x41, 0x53, 0xF8, 0x37, 0x18, 0x44, 0x24, 0x4F, 0x85, 0x33, 0x38, 0xFF, 0xC7, 0x29,
0x55, 0x42,
        0x87, 0x00, 0x01, 0x08,

        // ACPI_NOTIFY_POWER_LEVEL_D1
        // Notify event D1 - Power State P0
        // GUID {93263095-DA5F-46a0-8B1C-DB2F1F7D66AF}
        0x95, 0x30, 0x26, 0x93, 0x5F, 0xDA, 0xA0, 0x46, 0x8B, 0x1C, 0xDB, 0x2F, 0x1F, 0x7D,
0x66, 0xAF,
        0xD1, 0x00, 0x01, 0x08,

        // ACPI_NOTIFY_POWER_LEVEL_D2
        // Notify event D2 - Power State P1
        // GUID {BE907006-D603-4714-9AE2-9DBA8997A805}
        0x06, 0x70, 0x90, 0xBE, 0x03, 0xD6, 0x14, 0x47, 0x9A, 0xE2, 0x9D, 0xBA, 0x89, 0x97,
0xA8, 0x05,
        0xD2, 0x00, 0x01, 0x08,

        // ACPI_NOTIFY_POWER_LEVEL_D3
        // Notify event D3 - Power State P2
        // GUID {21D355E1-498D-4c1f-AE01-05CEF8DD053F}
        0xE1, 0x55, 0xD3, 0x21, 0x8D, 0x49, 0x1F, 0x4C, 0xAE, 0x01, 0x05, 0xCE, 0xF8, 0xDD,
0x05, 0x3F,
        0xD3, 0x00, 0x01, 0x08,

        // ACPI_NOTIFY_POWER_LEVEL_D4
        // Notify event D4 - Power State P3
        // GUID {EDBCEDF1-BBC9-420e-A3C0-322B1D37C3AA}
        0xF1, 0xED, 0xBC, 0xED, 0xC9, 0xBB, 0x0E, 0x42, 0xA3, 0xC0, 0x32, 0x2B, 0x1D, 0x37,
0xC3, 0xAA,
        0xD4, 0x00, 0x01, 0x08,

        // ACPI_NOTIFY_POWER_LEVEL_D5
        // Notify event D5 - Power State P4
        // GUID {377E84E1-13DE-41fb-A096-0F08528BA689}
        0xE1, 0x84, 0x7E, 0x37, 0xDE, 0x13, 0xFB, 0x41, 0xA0, 0x96, 0x0F, 0x08, 0x52, 0x8B,
0xA6, 0x89,
        0xD5, 0x00, 0x01, 0x08
    })
```

```
// Method Execution
// MXM Native Methods are called via WMMX Index
// ONLY include the methods that you actually have !
    Method(WMMX, 3)
    {
        CreateDWordField(Arg2, 0, FUNC)      // Get the function name
        If (LEqual(FUNC, 0x4D53445F))        // "_DSM"
        {
            If (LGreaterEqual(SizeOf(Arg2), 28))
            {
                CreateField(Arg2, 32, 128, MUID)
                CreateDWordField(Arg2, 20, REVI)
                CreateDWordField(Arg2, 24, SFNC)
                CreateDWordField(Arg2, 28, ARGD)
                If (LEqual(And(Arg1, 0xFF)), DGPU_SCOPE.GBUS())) {
                    Return(DGPU_SCOPE._DSM(MUID, REVI, SFNC, ARGD))
                }
                If (LEqual(And(Arg1, 0xFF)), DGPU2_SCOPE.GBUS())) {
```

```
                    Return(DGPU2_SCOPE._DSM(MUID, REVI, SFNC, ARGD))
                }
            }
        }
        Return (0)
    }


// Compiled form of the associated MOF declaration
Name(WQXM, Buffer() {
0x46,0x4F,0x4D,0x42,0x01,0x00,0x00,0x00,0xCC,0x05,0x00,0x00,0xFC,0x30,0x00,0x00,
0x44,0x53,0x00,0x01,0x1A,0x7D,0xDA,0x54,0x18,0xD2,0x97,0x00,0x01,0x06,0x18,0x42,
0x10,0x19,0x10,0x8A,0xE6,0x80,0x42,0x04,0x92,0x43,0xA4,0x30,0x30,0x28,0x0B,0x20,
0x86,0x90,0x0B,0x26,0x26,0x40,0x04,0x84,0xBC,0x0A,0xB0,0x29,0xC0,0x24,0x88,0xFA,
0xF7,0x87,0x28,0x09,0x0E,0x25,0x04,0x42,0x12,0x05,0x98,0x17,0xA0,0x5B,0x80,0x61,
0x01,0xB6,0x05,0x98,0x16,0xE0,0x18,0x92,0x4A,0x03,0xA7,0x04,0x96,0x02,0x21,0xA1,
0x02,0x94,0x0B,0xF0,0x2D,0x40,0x3B,0xA2,0x24,0x0B,0xB0,0x0C,0x23,0x02,0x8F,0x82,
0xA1,0x71,0x68,0xEC,0x30,0x2C,0x13,0x4C,0x83,0x38,0x8C,0xB2,0x91,0x45,0x60,0xDC,
0x4E,0x05,0xC8,0x15,0x20,0x4C,0x80,0x78,0x54,0x61,0x34,0x07,0x45,0xE0,0x42,0x63,
0x46,0x69,0x8F,0x02,0xAC,0x8E,0x42,0x93,0x4A,0x60,0xF7,0x02,0x34,0x0A,0xD0,0x26,
0xC0,0xA1,0x00,0x85,0x02,0xAC,0x61,0xC8,0x19,0x84,0x6C,0x61,0x0B,0xA3,0x41,0x01,
0x16,0x51,0x34,0x82,0xB3,0xA8,0x78,0x4E,0x42,0x09,0x68,0x6F,0x08,0xC2,0x33,0x66,
0x63,0x78,0x42,0x0C,0x52,0x19,0x86,0x20,0xE2,0x46,0x38,0x96,0xF6,0x07,0x41,0xE4,
0x2B,0x18,0x37,0x8B,0x34,0x1A,0xD4,0x58,0x13,0x1C,0xBB,0x47,0x73,0xC2,0x9D,0x0B,
0x90,0x3E,0x37,0x81,0x1C,0xDD,0x69,0xD4,0x39,0x68,0x32,0x3C,0x86,0x95,0xE0,0x3F,
0xC0,0xA7,0x00,0xBC,0x6B,0x40,0x4D,0xFF,0xE0,0x99,0x20,0x38,0xD4,0x10,0x3D,0xEA,
0x70,0x27,0x70,0x5E,0x47,0xC2,0x20,0x8E,0xE8,0xB8,0xB1,0xB3,0x3A,0x99,0x83,0x2E,
0x55,0x80,0xD9,0x03,0x80,0x06,0x97,0xE0,0xB4,0xCF,0x24,0xF4,0x7B,0xC0,0xF9,0xF4,
0x3C,0x36,0x36,0x08,0xD4,0xC8,0xFC,0xFF,0x87,0xF6,0x20,0x4F,0x2B,0x66,0xC8,0x67,
0x81,0xC3,0x62,0x62,0x21,0xB4,0x49,0x8D,0x07,0x08,0x1C,0x00,0x1E,0x0D,0x22,0xBC,
0x19,0x78,0xBE,0x26,0x18,0x14,0x42,0x4E,0xC6,0x83,0x12,0x73,0x3E,0x20,0x73,0x09,
0xF1,0x10,0x70,0x0C,0x31,0x82,0x9E,0x51,0xD1,0xC8,0x9A,0x88,0xCF,0x00,0x1E,0xB7,
0x65,0x81,0x50,0x02,0x03,0xFB,0xDC,0xF0,0x0E,0x61,0x58,0xDC,0x68,0x02,0xFB,0x7C,
0x61,0x81,0x83,0x42,0xC1,0xFA,0xAE,0x10,0xF3,0x79,0xE0,0x38,0x0F,0xD8,0xC2,0x27,
0x4A,0x80,0x0F,0xC9,0xA3,0x81,0x37,0x48,0xF8,0xD6,0x8F,0x00,0x04,0xF8,0x89,0xC1,
0x04,0x96,0x07,0x8C,0x1E,0xA5,0xFD,0x0A,0x40,0x08,0xFE,0x62,0xF1,0x14,0xF0,0x3A,
0x10,0xE1,0x58,0x60,0x8A,0x1C,0x1B,0x1A,0x9E,0x1F,0x1C,0xA2,0x47,0x3D,0x97,0xF0,
0x47,0x13,0xE5,0x14,0x0E,0xC7,0x47,0x0E,0x23,0x84,0x7F,0xD2,0x78,0xF2,0xB0,0xE6,
0x3B,0x80,0xA6,0xF4,0x16,0xF0,0xFE,0xE0,0x11,0x60,0x04,0x48,0x38,0x9A,0x60,0x22,
0x9F,0x4E,0x40,0xF5,0xFF,0x3F,0x9D,0x00,0xA6,0xA6,0x1C,0xF4,0xED,0xE4,0x29,0x22,
0x50,0x94,0x20,0x2F,0x01,0x6F,0x25,0x41,0x22,0xBC,0x97,0x3C,0x99,0x3C,0x9D,0x58,
0xDC,0x29,0x45,0x18,0x51,0x8E,0x3E,0x50,0x98,0x17,0x83,0xF7,0x13,0x43,0x06,0x09,
0xEB,0x79,0xC4,0x8A,0x13,0x30,0x58,0x94,0x37,0x94,0xA7,0x13,0x16,0xF2,0xA1,0x21,
0x0D,0xA7,0x13,0x80,0x02,0xFF,0xFF,0xD3,0x09,0xEC,0xA8,0xA7,0x13,0xF4,0x70,0x7D,
0xCA,0xC0,0x20,0x1F,0x90,0x4F,0x1D,0x3E,0x07,0x9C,0x86,0x8F,0x27,0x18,0x51,0xC7,
0x13,0xD4,0xE9,0xC1,0xC7,0x13,0x76,0x3E,0x38,0x8B,0xC7,0x00,0x9F,0x4E,0x30,0xF7,
0x04,0x9F,0x4E,0xC0,0x36,0x1A,0x18,0xC7,0x13,0xF0,0x1D,0x5D,0x70,0xC7,0x13,0x30,
0x0C,0x06,0xC6,0xF9,0x04,0x78,0x8C,0x17,0x77,0x0C,0xF0,0x10,0xF8,0x00,0x1E,0x37,
0x4E,0xCF,0x4A,0x27,0x85,0x3C,0x89,0xF0,0x41,0x61,0x0E,0x2C,0xC0,0xE1,0xFF,0x7F,
0x60,0x01,0x4C,0xDD,0x0F,0x4E,0xF9,0x79,0xC5,0x17,0x16,0x23,0xBF,0xA3,0xBC,0xA8,
0xC4,0x38,0xF0,0x77,0x95,0x28,0x1E,0xC1,0x9B,0x4A,0xB4,0x60,0x0F,0x2C,0x3E,0xAE,
0xF8,0x88,0x66,0xBC,0x58,0x87,0xF1,0xEA,0x62,0xA4,0x28,0x31,0xE3,0xBD,0xAC,0x19,
0xE2,0x81,0x05,0x60,0xC3,0xFF,0xFF,0xC0,0x02,0x4C,0xAE,0x41,0x0F,0x2C,0xC0,0x78,
0x34,0xEC,0xC0,0x02,0x4C,0x07,0xC3,0x0E,0x2C,0x80,0xCB,0xFF,0xFF,0x81,0x05,0x30,
0x75,0x20,0xF5,0x5D,0x25,0xC6,0x33,0x29,0x43,0x78,0x51,0x79,0x4F,0xF3,0x81,0x8D,
0x5F,0x0D,0x7C,0x30,0x78,0x5B,0x33,0x48,0xA4,0x07,0x16,0x5F,0x5D,0x0C,0x11,0x34,
0x62,0x8C,0xA8,0x0F,0x2C,0x0F,0x6E,0xC6,0x89,0xF1,0xC0,0x02,0xB0,0xE1,0xFF,0x7F,
0x60,0x01,0x26,0x57,0xA1,0x07,0x16,0x60,0x3C,0x1A,0x76,0x60,0x01,0xA6,0x83,0x61,
0x07,0x16,0xC0,0xE5,0xFF,0xFF,0xC0,0x06,0xB8,0xBA,0xA2,0x3C,0xA1,0xFA,0x9E,0x12,
0xE5,0x15,0xC5,0xB7,0x15,0x83,0xF8,0xB6,0xE2,0x83,0x9A,0xCF,0x06,0x0F,0xAC,0x06,
0x64,0x07,0x56,0x43,0x3C,0xB5,0x1A,0xE1,0x09,0x20,0x78,0xB8,0x13,0x09,0x16,0xE1,
0xD1,0xC5,0x07,0x16,0x80,0x0D,0xFF,0xFF,0x03,0x0B,0x30,0xB9,0x09,0x3C,0xB0,0x00,
0xE3,0xD1,0xB0,0x03,0x0B,0x30,0x1D,0x0C,0x3B,0xB0,0x00,0x2E,0xFF,0xFF,0x07,0x16,
0xC0,0xD4,0x71,0xCD,0x87,0x82,0x17,0xA2,0x18,0x81,0x7D,0x5B,0x33,0xDA,0xBB,0xBA,
0x0F,0xAC,0x7C,0x1C,0xC1,0x1E,0x58,0x7C,0xC1,0x78,0x6A,0x39,0xFC,0x78,0xBE,0xB2,
0x1A,0xEF,0x69,0xCD,0xC7,0x83,0x38,0x11,0x23,0x04,0x7E,0x60,0x03,0xF8,0xF0,0xFF,
0x3F,0xB0,0x00,0x8F,0x83,0xC0,0x03,0x2B,0x30,0x1E,0x0D,0x3F,0xB0,0x00,0xCF,0xC1,
0xF0,0x03,0x0B,0xE0,0xF2,0xFF,0x7F,0x60,0x01,0x4C,0xDD,0x51,0x0D,0x1A,0xF6,0x19,
0x25,0x44,0x84,0x67,0x94,0x17,0x15,0x63,0x44,0x79,0x56,0xF3,0x08,0x62,0x3D,0x17,
0x3C,0xB0,0x19,0xF1,0xE0,0x83,0x3E,0x1E,0x44,0xF1,0x85,0xC5,0x38,0x2F,0x2D,0x51,
0x22,0x04,0x8C,0xE2,0x73,0xAB,0x0F,0x2C,0x00,0x1B,0xFE,0xFF,0x07,0x16,0x60,0x72,
0x0E,0x61,0x07,0x16,0x60,0x3B,0x1A,0x76,0x60,0x01,0xA6,0x03,0x60,0x07,0x16,0xC0,
```

```
0xE5,0xFF,0xFF,0xC0,0x02,0x98,0x3A,0x0F,0xBC,0xA8,0x84,0x79,0x42,0x78,0x44,0xF5,
0x31,0xDD,0xB7,0x15,0x83,0x3F,0xA5,0x84,0x8A,0xE2,0x71,0x44,0x7A,0x0F,0x88,0xF2,
0xBE,0xE2,0x41,0xC4,0xF3,0x59,0xC5,0x27,0x02,0x9F,0x5E,0x0D,0x19,0x21,0x90,0x8F,
0xEC,0x3E,0xB0,0x00,0x6C,0xF8,0xFF,0x1F,0x58,0x80,0xC9,0x39,0xE0,0x81,0x05,0x18,
0x8F,0x86,0x1D,0x58,0x80,0xE9,0x60,0xD8,0x81,0x05,0x70,0xF9,0xFF,0x3F,0xB0,0x00,
0xA6,0x86,0xFB,0x90,0xE6,0x5B,0x80,0x41,0xDE,0x52,0x7D,0x51,0x89,0x13,0xC8,0xF0,
0x0F,0x2C,0xBE,0x1A,0x78,0x1A,0x0F,0x2C,0x3E,0x19,0x84,0x7A,0x65,0xF5,0x5D,0xC1,
0x08,0xEF,0xAC,0x3E,0x1E,0xF8,0xC0,0x16,0x2B,0x46,0xBC,0x08,0x0F,0x2C,0x00,0x1B,
0xFE,0xFF,0x07,0x16,0x60,0x72,0x59,0xF0,0x81,0x05,0xF8,0x8E,0x86,0x1D,0x58,0x80,
0xE9,0x60,0xD8,0x81,0x05,0x70,0xF9,0xFF,0x3F,0xB0,0x00,0xA6,0x4E,0x28,0x46,0x38,
0xDF,0x77,0xB4,0xC8,0x0F,0x6C,0x0C,0x22,0x54,0xBC,0x30,0x0F,0x2C,0xBE,0xAF,0x59,
0xF1,0x81,0x85,0xCC,0xC1,0xF7,0x56,0x83,0xBC,0xB9,0x84,0x88,0xFB,0xB6,0xF6,0x86,
0xF0,0xD6,0x6E,0xB4,0x58,0xCF,0xAC,0x3E,0xB0,0x00,0x6C,0xF8,0xFF,0x1F,0x58,0x80,
0xC9,0x51,0xE0,0x81,0x05,0x18,0x8F,0x86,0x1D,0x58,0x80,0xE9,0x60,0xD8,0x81,0x05,
0x70,0xF9,0xFF,0x3F,0xB0,0x03,0xAE,0x0E,0x08,0xC7,0x1B,0xF7,0x25,0xC5,0x57,0x62,
0x5F,0xD4,0x3C,0xF0,0x08,0xAF,0xAC,0xEC,0xBC,0x72,0x04,0x0F,0xEC,0xC6,0x7B,0x5B,
0x09,0x16,0xCA,0x77,0x17,0xDF,0x56,0x8C,0x1A,0x29,0xF0,0xA3,0x9B,0xF1,0x9F,0xE0,
0x7D,0x60,0x01,0xD8,0xF0,0xFF,0x3F,0xB0,0x00,0x8F,0xA3,0xB0,0x0F,0x2C,0xC0,0x77,
0x34,0xFC,0xC0,0x02,0x3C,0x07,0xC3,0x0F,0x2C,0x80,0xCB,0xFF,0xFF,0x81,0x0D,0x70,
0x75,0x40,0x8B,0x1A,0x27,0xC6,0x1B,0xC1,0x9B,0xC5,0x8B,0x9A,0xAF,0xA9,0x3E,0xB3,
0xF9,0xC0,0xE2,0x6B,0x80,0xA1,0x8C,0xF6,0xE0,0x66,0x88,0x78,0xBE,0xAB,0x18,0xFB,
0xC9,0xCD,0x08,0x91,0xC2,0xBC,0x03,0xBC,0xB4,0xF8,0xC0,0x02,0xB0,0xE1,0xFF,0x7F,
0x60,0x03,0x26,0x77,0x41,0x1F,0x58,0x80,0xED,0x64,0xD8,0x81,0x05,0x98,0x0E,0x86,
0x1F,0x58,0x00,0x27,0x0A,0x6D,0xFA,0xD4,0x68,0xD4,0xAA,0x41,0x99,0x1A,0x65,0x1A,
0xD4,0xEA,0x53,0xA9,0x31,0x63,0xC7,0x14,0x07,0x7C,0x08,0xE8,0x44,0x60,0x79,0xCF,
0x06,0x81,0x58,0x06,0x85,0x40,0x2C,0xFD,0xD5,0x22,0x10,0x07,0x07,0xD1,0xFF,0x1F,
0xC4,0x11,0x9F,0x1C,0x02,0x23,0x26,0xC0,0x28,0x08,0x8D,0xA8,0x02,0x8C,0xDA,0x00,
0xA3,0x20,0x34,0xA2,0x0E,0x30,0x6A,0x04,0x8C,0x82,0xD0,0x88,0x4A,0xC0,0xA8,0x15,
0x30,0x0A,0x42,0x23,0x6A,0x01,0xA3,0x66,0xC0,0x28,0x08,0x8D,0xA8,0x06,0x8C,0xDA,
0x01,0xA3,0x20,0x34,0xA2,0x1E,0x30,0x6A,0x08,0x8C,0x82,0xD0,0x88,0x8A,0xC0,0xA8,
0x25,0x30,0x0A,0x42,0x23,0x6A,0x02,0xA3,0xA6,0xC0,0x28,0x08,0x8D,0xA8,0x0A,0x8C,
0xBE,0x59,0x82,0x31,0x10,0x1A,0x51,0x17,0x08,0xFB,0xFF,0x03
))

// Associated MOF declaration

[WMI, Dynamic, Provider("WMIProv"),
 Locale("MS\\0x409"),
 GUID("{F6CB5C3C-9CAE-4ebd-B577-931EA32A2CC0}")]
class MXM20Method
{
    [key, read]
    String InstanceName;
    [read] Boolean Active;
    [WmiMethodId(1),
     Description("MXM20Method")
    ] uint32 MXM20Method;
};

[WMI, Dynamic, Provider("WMIProv"),
 Locale("MS\\0x409"),
 GUID("{E06BDE62-EE75-48F4-A583-B23E69ABF891}")]
class MXM20Event80 : WMIEvent
{
    [key, read]
    String InstanceName;
    [read] Boolean Active;
    [WmiDataId(1),
     Description("MXM20Event80")
    ] uint32 MXM20Event80;
};

[WMI, Dynamic, Provider("WMIProv"),
 Locale("MS\\0x409"),
 GUID("{3ADEBD0F-0C5F-46ED-AB2E-04962B4FDCBC}")]
class MXM20Event81 : WMIEvent
{
    [key, read]
    String InstanceName;
    [read] Boolean Active;
    [WmiDataId(1),
     Description("MXM20Event81")
```

```
    ] uint32 MXM20Event81;
};

[WMI, Dynamic, Provider("WMIProv"),
 Locale("MS\\0x409"),
 GUID("{1E519311-3E75-4208-B05E-EBE17E3FF41F}")]
class MXM20Event86 : WMIEvent
{
    [key, read]
    String InstanceName;
    [read] Boolean Active;
    [WmiDataId(1),
     Description("MXM20Event86")
    ] uint32 MXM20Event86;
};

[WMI, Dynamic, Provider("WMIProv"),
 Locale("MS\\0x409"),
 GUID("{37F85341-4418-4F24-8533-38FFC7295542}")]
class MXM20Event87 : WMIEvent
{
    [key, read]
    String InstanceName;
    [read] Boolean Active;
    [WmiDataId(1),
     Description("MXM20Event87")
    ] uint32 MXM20Event87;
};

[WMI, Dynamic, Provider("WMIProv"),
 Locale("MS\\0x409"),
 GUID("{921A2F40-0DC4-402d-AC18-B48444EF9ED2}")]
class MXM20EventD0 : WMIEvent
{
    [key, read]
    String InstanceName;
    [read] Boolean Active;
    [WmiDataId(1),
     Description("MXM20EventD0")
    ] uint32 MXM20EventD0;
};

[WMI, Dynamic, Provider("WMIProv"),
 Locale("MS\\0x409"),
 GUID("{93263095-DA5F-46a0-8B1C-DB2F1F7D66AF}")]
class MXM20EventD1 : WMIEvent
{
    [key, read]
    String InstanceName;
    [read] Boolean Active;
    [WmiDataId(1),
     Description("MXM20EventD1")
    ] uint32 MXM20EventD1;
};

[WMI, Dynamic, Provider("WMIProv"),
 Locale("MS\\0x409"),
 GUID("{BE907006-D603-4714-9AE2-9DBA8997A805}")]
class MXM20EventD2 : WMIEvent
{
    [key, read]
    String InstanceName;
    [read] Boolean Active;
    [WmiDataId(1),
     Description("MXM20EventD2")
    ] uint32 MXM20EventD2;
};

[WMI, Dynamic, Provider("WMIProv"),
 Locale("MS\\0x409"),
 GUID("{21D355E1-498D-4c1f-AE01-05CEF8DD053F}")]
class MXM20EventD3 : WMIEvent
```

```
{
    [key, read]
    String InstanceName;
    [read] Boolean Active;
    [WmiDataId(1),
     Description("MXM20EventD3")
    ] uint32 MXM20EventD3;
};

[WMI, Dynamic, Provider("WMIProv"),
 Locale("MS\\0x409"),
 GUID("{EDBCEDF1-BBC9-420e-A3C0-322B1D37C3AA}")]
class MXM20EventD4 : WMIEvent
{
    [key, read]
    String InstanceName;
    [read] Boolean Active;
    [WmiDataId(1),
     Description("MXM20EventD4")
    ] uint32 MXM20EventD4;
};

[WMI, Dynamic, Provider("WMIProv"),
 Locale("MS\\0x409"),
 GUID("{377E84E1-13DE-41fb-A096-0F08528BA689}")]
class MXM20EventD5 : WMIEvent
{
    [key, read]
    String InstanceName;
    [read] Boolean Active;
    [WmiDataId(1),
     Description("MXM20EventD5")
    ] uint32 MXM20EventD5;
};
```

3.3.2. _DSM (Device Specific Methods)

In order to prevent collision of method names and allow future expansion, the standard _DSM method is exposed under each GPU to provide support for MXM sub-functions.

Arguments

***Arg0: GUID* (16 Byte Buffer)**

MXM_DSM_GUID {4004A400-917D-4cf2-B89C-79B62FD55665}

Arg1: Revision ID (DWord Integer)

The requested interface revision ID expressed in Binary Coded Decimal form where the Lower bytes are the Major and Minor version respectively. If '0' then the default, highest level of interface is supported.

MXM_REVISION_ID	0x00000300	Interface revision level

Arg2: FunctionCode (DWord Integer)

The functions supported through this interface method include:

MXM_FUNC_MXSS	0x00000000	Supported Sub-Functions
MXM_FUNC_MXMI	0x00000018	Platform MXM Capabilities
MXM_FUNC_MXMS	0x00000010	Get the MXM Structure

MXM_FUNC_MXPP	0x00000004	Get/Set Platform Policies (see Chapter 4)
MXM_FUNC_MXDP	0x00000005	Get/Set Display Config (see Chapter 4)
MXM_FUNC_MDTL	0x00000006	Get Display Toggle List (see Chapter 4)
MXM_FUNC_EVENTLIST	0x00000012	Flexible Event Notifiers (see Chapter 4)
MXM_FUNC_MXCB	0x00000019	Query/Call System Callbacks (see Chapter 4)

Arg3: Arguments (Package)

Following, see description of arguments and return values per sub-function.

On Return

Sub-function 0 and unsupported function calls always returns a buffer. Other sub-functions may return a buffer or a package as defined in the function.

When a single DWord is returned the following values have special meaning, controlled by reserved Bit31 as follows:

Status (DWord Integer)

MXM_ERROR_SUCCESS	0x00000000	Success
MXM_ERROR_UNSPECIFIED	0x80000001	Generic unspecified error code
MXM_ERROR_UNSUPPORTED	0x80000002	*FunctionCode* or *SubFunctionCode* not supported by this system

Example

The following is hypothetical stripped-down example.

```
// _DSM Device Specific Method
//      Arg0: UUID Unique function identifier
//      Arg1: Integer Revision Level
//      Arg2: Integer Function Index (0 = Return Supported Functions)
//      Arg3: Package Parameters

Method (_DSM, 4, NotSerialized)
{
    // Switch based on which unique function identifier was passed in
    // Note:ToUUID("{4004A400-917D-4cf2-B89C-79B62FD55665}") is not well supported
    If (LEqual(UCMP(Arg0, Buffer(){0x00, 0xA4, 0x04, 0x40, 0x7D, 0x91, 0xF2, 0x4C,
    0xB8, 0x9C, 0x79, 0xB6, 0x2F, 0xD5, 0x56, 0x65 })))
    {
        Switch (Arg2)
        {
            //
            // Function 0: MXM_FUNC_MXSS
            case (0)
            {
                // Sub-Functions 0, 16, 24 are supported
                Return(Buffer(){0x01010001})
            }
            //
            // Function 24: MXM_FUNC_MXMI
            case (24)
            {
                Return(Buffer(){0x30})
            }
            //
            // Function 16: MXM_FUNC_MXMS
            case (16)
            {
                // Replace with Platform SIS info
                If (LEqual (Arg1, 0x30)) {
                    Return(MXM3)
                }
            }
        //
        // ... etc
        //
        }
    }
}
```


3.3.2.1. MXM_FUNC_MXSS – Return Supported Sub-Functions

Sub-function 0 returns the interface sub-functions supported by the method.

Arguments

None

On Return

The return buffer has a bit set for every sub-function supported.

Table 3-1. MXM_FUNC_MXSS Return Buffer

Bits	Definition
31:26	Reserved. Must be zero
25	1 = MXM_FUNC_MXCB, Get System Callbacks
24	1 = MXM_FUNC_MXMI, Specification Support Level
23:19	Reserved. Must be zero.
18	1 = MXM_FUNC_EVENTLIST, Reassign event notifiers
17	Reserved. Must be zero.
16	1 = MXM_FUNC_MXMS, Get platform MXM Structure
15:7	Reserved. Must be zero.
6	1 = MXM_FUNC_MDTL, Get Display Toggle List
5	1 = MXM_FUNC_MXDP, Get Display & Hot-Key information
4	1 = MXM_FUNC_MXPP, Get/Set the platform policy settings
3:1	Reserved. Must be zero.
0	1 = MXM_FUNC_MXSS, Supported Sub-Functions (this call)

Example

```
// Function 0: MXM_FUNC_MXSS
case (0) {
    // Sub-Functions 0, 16, 24 are supported
    Return(Buffer(){0x01010001})
}
```


3.3.2.2. MXM_FUNC_MXMI – Return Specification Support Level

Sub-function 24 is a required method which returns information about the level of the MXM software specification that the system supports. Calling this method with an argument (Arg1) of "0" shall always return the default or highest level supported. If additional versions of the MXM software interface are supported, the method shall indicate the specific version in the return value when queried with that version value as an input argument.

Arguments

None. Note that Arg1 already contains the requested revision level.

On Return

The return buffer contains the same version level as input Arg1 if it is supported. If Arg1 was 0, returns the highest supported version level.

Table 3-2. MXM_FUNC_MXMI Return Buffer

Bits	Definition
7:0	Supported Interface Version Requested version level in BCD format.
31:8	Reserved. Must be zero

Example

```
// Function 24: MXM_FUNC_MXMI
case (24) {
   Return(Buffer(){0x30})
}
```

3.3.2.3. MXM_FUNC_MXMS – Return MXM Structure

Sub-function 16 is a required method which returns the size of the MXM structure, and the structure (or a segment up to 4 Kbytes if the structure is larger than 4 KB). If supported a system may provide different version MXM software structures depending on the caller requested version.

Arguments

The requested MXM interface version level.

Table 3-3. MXM_FUNC_MXMS Structure Identifier

Bits	Definition
7:0	Page Offset 4KB Page Offset into the object i.e. 0 = first 4 KB page, 1 = second 4 KB page, etc
15:8	Reserved. Must be zero
31:16	MXM Structure Identifier Requested version level of MXM structure in BCD format (for example: 0030h = 3.0, etc.). If zero, then the structure version indicated in the return of MXMI will be supplied. To obtain an additional vendor specific data block, specify an identifier in the range 0x80-0x8F. If the system does not contain such a data block, it should return an error. Details for any such OEM or graphics vendor specific data blocks are outside the scope of this document.

On Return

The MXM structure as a Buffer.

Example

```
Name(MXM3, Buffer() {...}) // An MXM v 3.0 structure

// Function 16: MXM_FUNC_MXMS
case (16) {
    If (LEqual (Arg1, 0x30)) {
        Return(MXM3)
    }
}
```

4 Additional MXM System Interfaces

Minimum interface requirements are outlined in Chapter 3. The following interfaces may be required depending on system features.

4.1. Additional MXM INT 15H System BIOS Callbacks

4.1.1. Function 2 – Return a Pointer to the EDID Structure for the Internal Panel

This is a required function for systems containing an internal flat panel without an EDID on DDC/Aux. This function allows the VBIOS to receive a pointer to the EDID structure that should be used for the internal flat panel in the system. This is required in cases where the panel being used does not have an EDID structure which can be read through DDC/Aux lines. This structure resides in the SBIOS ROM area or in another memory location that is accessible in real mode during video POST.

Entry:

AX = 5F80h

BL = 02h

BH = FFh

EBX[16:31] = Adapter ID (see above)

EDX = Display device identifier (32-bit value as used in _DOD)

Return:

AX = 005Fh to indicate that the system bios supports this function

BL = EDID structures returned

03 = 128byte EDID 1.4 structure

04 = 128byte EDID 1.4 followed by a 128byte DI-EXT block

ES:DI = Pointer to the EDID structure in real mode memory (< 1MB)

The EDID structure shall comply with VESA E-EDID multi-block format. The first block of 128 bytes shall be a VESA 1.4 EDID. A digital extension in VESA DI-EXT format may optionally also be included. The VBIOS will attempt to read the EDID using this Int15h callback first, then attempt to read the panel EDID via DDC, only if this function fails. If present, EDID information shall override MXM structure information. For example, as relates to link width or pixel depth.

4.1.2. Function 3 – Select Output Device Channel

This is an optional function, which is only required if a multiplexer is used for controlling DDC or display outputs. The function is used when performing a display switch to an output device that is controlled by a multiplexer connected to a GPIO. In order to allow the SBIOS to properly set up selection of the DDC port and/or the data output pins, the VBIOS will call this function before attempting to access the DDC port for the device or to set up and enable output to the device.

When selecting multiplexed DDC lanes, the VBIOS will call to acquire, and call again to release when the channel is no longer needed. If DDC lanes are shared between displays the SBIOS is responsible for creating a mutex to co-ordinate between VBIOS (thru Int15h/EFI) and driver (thru MXMX) access. If the VBIOS has acquired the mutex first then a simultaneous attempt thru MXMX should fail until the mutex is released, and vice-versa VBIOS should fail if the channel was already acquired through MXMX.

Entry:

AX = 5F80h

BL = 03h

BH = FFh

EBX[16:31] = Adapter ID (see above)

CL = Selection

0 – Acquire shared Display DDC

1 – Display Output

3 – Both

4 – Release shared Display DDC

EDX = Display device identifier (32-bit value as used in _DOD)

Exit:

AX = 005Fh	to indicate that the SBIOS supports this function
BL	For CL=0 the SBIOS shall return a 0h if the mutex was successfully acquired. When non-zero the mutex was not aquired

4.1.3. Function 4 – Boot Message

This is an optional function which returns a pointer to a sign-on message which VBIOS will display during POST, or indicate the VBIOS should not display any boot message. No delay is specifiable, it is up to the OEM SBIOS to control execution, and thereby the duration the screen retains the message.

Entry:

AX = 5F80h

BL = 04h

Return:

AX = 005Fh	To indicate that the system bios supports this function
BL = Mode	0 = the Pointer points to a (zero-terminated) Sign-On Text String. Non-zero = Reserved.
ES:DI = Pointer	String in real mode memory (< 1MB). A zero length string indicates the normal sign-on message should be suppressed e.g. to support a graphical splash screen.

4.1.4. Function 7 – Return a Pointer to the VBIOS Image for ROM-Less Adapters

This is an optional function, required if the system supports ROM-less MXM modules. Access to the VBIOS image is required for test purposes prior to the load of an operating system which supports ACPI.

Entry:

AX = 5F80h

BL = 07h

BH = FFh

EBX[16:31] = Adapter ID (see above)

Return:

AX = 005Fh to indicate that the system bios supports this function

EDI = Physical Memory offset to the 128K aperture containing the selected VBIOS image.

The SBIOS is responsible for reserving physical memory (for example, via ACPI) for the 128 K Aperture to pass VBIOS. The caller is responsible for mapping the physical address to linear before access. The VBIOS Image returned in the aperture follows the PCI standard for Option ROM's including the option ROM structure to index additional objects within the image.

4.1.5. Function 8 – Check Availability of Output Device

This is an optional function, which is only required if one or more of the output devices listed in the MXM output device structures are not available under certain system conditions. Examples include display connectors available on a docking station, which are not present if the system is not currently docked. When detecting the presence of an output device, the VBIOS will first call this function with the appropriate display device identifier. If the function returns BX=0, the connector is assumed to be present and the VBIOS will attempt to detect whether a display is present on the connector. If the function returns a value in BX other than 0, the VBIOS will know the connector is not currently available. It will then skip this detect attempt. If the function is not implemented (AX returns a value other than 0x005F) the connector is assumed to be available.

The success or failure of a previous attempt is not stored. That is, at every display device detect the VBIOS will again call the SBIOS to check for availability of the appropriate connector. If the docking state (for example) has changed since the last detect, the SBIOS can modify its return values appropriately.

Entry:

AX = 5F80h

BL = 08h

BH = FFh

EBX[16:31] = Adapter ID (see above)

EDX = Display device identifier (32-bit value as used in _DOD)

Exit:

AX = 005Fh	to indicate that the SBIOS supports this function
BL = Available	BL = 0 indicates the connector is currently available. BL = 1 indicates the connector is not currently available and no display detection should occur on it.

4.1.6. Function 9 – Identify Output Devices

This function returns the same list of output device identifiers as the ACPI _DOD method. It is used by the GPU software to map the list of display connectors the MXM module supports with those actually present on the system. It is required if the SBIOS implements any of the functions that require a display device identifier (functions 2, 3, or 8).

Entry:

AX = 5F80h

BL = 09h

BH = 00h

Exit:

AX = 005Fh	to indicate that the SBIOS supports this function

BL = Number of entries in table.

ES:DI = Pointer to the table. This is a table of 32-bit values identical to the buffer returned by _DOD (see section 4.3.10).

4.2. Additional MXM EFI System BIOS Callbacks

4.2.1. Returning the EDID Structure for the Internal Panel

The EDID for an integrated panel without an actual DDC connection should be made available using EFI_EDID_OVERRIDE_PROTOCOL, as described in the UEFI specification. This call uses a ChildHandle argument to describe the specific output device being referenced.

4.2.2. MxmSelectOutputDevice – Select Output Device Channel

Equivalent to Int15h function 3. Caller provides the handle of the target adapter, the handle of the display and the output setting.

```
typedef EFI_STATUS (EFIAPI *MXM_SELECT_OUTPUT_DEVICE)(
   IN struct _MXM3_EFI_INTERFACE  *This,
   IN    EFI_HANDLE  ControllerHandle,
   INOUT UCHAR16  usStructSize,
   IN    EFI_HANDLE  ChildHandle,
   IN    UCHAR8   ucOutputSetting
);
```

Where the value passed in ucOutputSetting may indicate one of the following actions:

0 – Acquire shared Display DDC

1 – Enable this Display Output

3 – Enable this Display - Both Output and DDC

4 – Release shared Display DDC

4.2.3. MxmCheckOutputDevice – Check Availability of Output Device

Equivalent to Int15h function 8. Caller provides the handle of the target adapter and the handle of the display.

```
typedef EFI_STATUS (EFIAPI *MXM_CHECK_OUTPUT_DEVICE)(
   IN struct _MXM3_EFI_INTERFACE  *This,
   IN    EFI_HANDLE  ControllerHandle,
   INOUT UCHAR16  usStructSize,
   IN    EFI_HANDLE  ChildHandle,
   OUT   UCHAR8   ucDisplayAvailable
);
```

The returned value ucDisplayAvailable will be one of the following values:

0 – Connector is currently available

1 – Connector is not currently available

4.2.4. Return a Pointer to the VBIOS image for ROM-less Adapters

For systems which support modules without ROMs (by including the VBIOS image in the system flash), the SBIOS must support the RomImage query as defined in the EFI_PCI_IO_PROTOCOL. Refer to the following ACPI discussion of _ROM usage for more details.

4.2.5. Identify Output Devices

The same ACPI identifier as described under _DOD (see Section 4.3.10) should be placed in the _ADR device path of the EFI_DEVICE_PATH_PROTOCOL.

4.3. Additional MXM ACPI Methods

4.3.1. Loading VBIOS image for ROM-less Adapters

On systems which support modules without ROMs by including the VBIOS image in the system flash, the _ROM method shall be supported allowing retrieval of portions of the VBIOS image necessary for run-time driver self-configuration. This requirement also covers systems with a ROM-less secondary adapter.

The stored image may contain more than the minimal Int10 VBIOS, and can also contain EFI BIOS, or other private blocks. In such cases the image will comply with the PCI specification for multiple section PCI Expansion ROMs.

4.3.2. ACPI Notification

Notify codes 0xD1 ~0xD5 are reserved for indicating power events P0 ~ P4.

As per ACPI specification Display Hot Plug Notification is signaled through Notify (VGA, 0x81) and Hot-Key through Notify (VGA, 0x80).

Note:	For proper operation under Windows XP these methods should be available under the WMI namespace as well as the graphics device.

4.3.3. Returning the EDID Structure for the LVDS Panel via ACPI

When using ACPI video extensions, the EDID for an integrated panel without an actual DDC connection must be made available by the system through the _DDC method of the display. For example, SB.PCI.VGA.LCD._DDC(0) using the namespace in the given example. Multiple block E-EDIDs are possible by using an EDID 1.4 structure together with extensions such as the DI-EXT block.

4.3.4. Retrieving the Backlight Control Settings for the LVDS Panel via ACPI

When using ACPI video extensions, the _BCL, _BCM, and _BQC methods shall be provided allowing the control necessary for backlight operation.

4.3.5. Check Availability of Output Device (Docking Stations and LID Display State)

In order to provide a single consistent method that works under all operating systems, the _DCS method inside the display device is used to indicate whether each connector is currently available. A

value of 0 indicates that the referenced connector is not available. This can be used to reflect the LID or docking state for a display.

For example to indicate availability of a CRT connector on the dock:

```
Device(CRT0, 0)}
    Name(ADR, CRT0_DOD)
    Method(_DCS, 0) {
        If (bDocked)
            Return( /* Any value with Bit 0 = 1 */ )
        Else
            Return(0)
    }
    // ... etc
}
```

For Internal Flat Panel devices the DCS method shall also indicate Lid state, for example:

```
Device(LCD0, 0)}
    Name(ADR, LCD0_DOD)
    Method(_DCS, 0) {
        If (bLidClosed)
            Return(0)
        Else
            Return( /* Any value with Bit 0 = 1 */ )
    }
    // ... etc
}
```

Any change in state shall be signaled via a Hot-Plug notify (VGA, 0x81).

If a display device is not to be enabled or selected then the system may also use the _DCS method to prevent the device from being included in the Hot-Key chain. If a display's availability will not be affected by system state (for example, always present when docked or undocked, lid open or closed) then _DCS need not be implemented for that display.

4.3.6. Selecting the Display Output via ACPI

If the MXM output device structure bit 34 indicates system methods are used for output switching then all necessary switching shall be performed inside MXDS in order to select the targeted display output. That is, the GPU software will call the MXDS method in order to request from the system that the output routing on the motherboard be enabled for that display.

Control of the display output *data* channel (DDC or Aux) is separate from the display output control. When selecting a multiplexed data channel, the GPU software will call the MXMX method (described in Section 4.3.9) for the appropriate display device.

4.3.7. _DSM (Device Specific Methods)

4.3.7.1. MXM_FUNC_MXPP – Platform Policy

Sub-function 4 gets or sets current system policy settings, for example as are saved in **Non-Volatile** storage from BIOS setup options. This sub-function is required only if any of these settings are maintained by the system BIOS.

Most fields have an associated update bit. When communicating to the SBIOS, the GPU driver sets the bit for any field it is providing a new setting for (for example, one that has been updated through an OS or user selection). Conversely, the SBIOS sets the bit for an associated field if it wishes to override the current GPU driver setting for that field. SBIOS shall clear this flag once any new settings are saved. This method would be evaluated at boot and when a notify (VGA, 0xEF) is signaled.

Arguments

The new policy settings to be saved in non-volatile storage:

Table 4-1. MXM_FUNC_MXPP Platform Policy

Bits	Definition
11:0	Boot Display Preference The user selected display combination, expressed in order of the display enumerated under _DOD. Zero indicates no preference, or automatic based on order of displays enumerated in DOD. Bit 0 : 1st Display ID enumerated Bit 1 : 2nd Display ID enumerated …etc
12	Update Boot Display Mask If set to a '1' this mask bit indicates the Boot display has been updated (SBIOS may wish to update it's non volatile storage)
14:13	Panel Scaling Preference The preferred flat panel scaling setting 0 – On: Auto Scaling (modes are scaled retaining aspect ratio) 1 – On: Force Scaling (modes are scaled to fill the screen) 2 – Off (no scaling)
15	Reserved. Must be zero
16	Update Scaling Preference Mask If set to a '1' this mask bit indicates Panel Scaling has been updated
20:17	Preferred TV Connector 0x00 = Automatic. Default is Composite (re-uses Y pin) 0x01 = Composite (re-uses CVBS pin) 0x02 = YPrPb Component

Bits	Definition
	0x03 = RGB Component 0x04 = S-Video (4 or 7 pin, uses C and Y) 0x09 = Japanese D-Connector 0x0F = Not specified. Determined at run time
21	Update TV Connector Mask If set to a '1' this mask bit indicates the TV connector has been updated
26:22	Preferred TV Format 0x00 = NTSC_M (US) 0x01 = NTSC_J (Japan) 0x02 = PAL_M (Brazilian format) 0x03 = PAL_BDGHI 0x04 = PAL_N (Paraguay and Uruguay format) 0x05 = PAL_NC (Argentina format) 0x06 = SECAM_L (France) 0x07 = Reserved for future use 0x08 = HD576i 0x09 = HD480i 0x0A = HD480p 0x0B = HD576p 0x0C = HD720p 0x0D = HD1080i 0x0E = HD1080p 0x1F = Not specified. Determined at run time
27	Update TV Preference Mask If set to a '1' this mask bit indicates the TV format has been updated
31:28	Reserved. Must be zero

On Return

The return buffer contains the platform policy status.

Table 4-2. MXM_FUNC_MXPP Return Buffer Policy Status

Bits	Definition
11:0	Boot Display Preference The system stored boot display combination. Same field format as previously described
12	Boot Display Override The setting overrides registry settings. This is cleared by updating the setting
14:13	Panel Scaling preference Same as previously described
15	Panel Scaling Override The setting overrides registry settings. This is cleared by updating the setting
19:16	Preferred TV Connector Same as previously described
20	TV connector Override The setting overrides registry settings. This is cleared by updating the setting
25:21	Preferred TV Format Same as previously described
26	TV Preference Override The setting overrides registry settings. This is cleared by updating the setting
31:27	Reserved. Must be zero

4.3.7.2. MXM_FUNC_MXDP – Display Status

Sub-function 5 sets or returns the current display detection status, hotkey toggle sequence, and also requires the implementation of the MDTL method. This sub-function is required only if the SBIOS expects to exercise control over the detection or selection of displays, such as for notebook display switch hotkeys. The GPU driver provides the current display status in the call, and in the return buffer the system provides current system status.

The SBIOS is responsible for collecting the current information and presenting through the return buffer in this interface. This method would be evaluated on a Notify (VGA, 0x81) in order to discover which display had been hot-plugged and on a Notify (VGA, 0x80) in response to a hotkey in order to discover the next display in a hotkey sequence. The SBIOS can also generate a reevaluation using the MXM event Notify (VGA, 0xF0).

Note: The SBIOS is only expected to provide hot-plug information for displays which it can detect and the GPU software cannot. For example, the lid state for the LVDS panel, or a proprietary load or pin based detect for a display which does not provide a hot-plug pin to the MXM module. Such connectors are identified in the Output Device Structure (bit 47).

Arguments

The latest display configuration

Table 4-3. MXM_FUNC_MXDP Display Status

Bits	Definition
11:0	Attached Displays The display device(s) detected as attached, expressed as a bit-field of display enumerated under _DOD. Zero indicates no active displays. Bit 0 : 1st Display ID enumerated Bit 1 : 2nd Display ID enumerated …etc
23:12	Active Displays The device(s) actively displaying content, expressed as a bit-field of display enumerated under _DOD. Zero indicates no active displays.
24	Display Mask If set to a '1' this mask bit indicates the Active (23:12) & Attached (11:0) fields contain significant info.
29:25	Next Combination Sequence Number The sequential index of the display combination in the Display toggle list (as described by MDTL), the first entry being index = '1'. A '0' value indicates a display combination not in the list. This field is passed after the next sequence in the toggle list is determined by the Driver.
30	Reserved. Must be zero
31	Next Combination Sequence Mask If set to a '1' this mask bit indicates the preceding display combination fields contain significant info.

On Return

The return buffer contains the status of events.

Table 4-4. MXM_FUNC_MXDP Return Buffer Status

Bits	Definition
3:0	Display ACPI Event 0 : No Hot-Key 1 : 0x80 Display Switch Hot Key 2 : 0x81 Display Hot-Plug (e.g. due to HPD, or Docking) or Lid Open/Close 3 : 0x86 Brightness Increase Hot-Key 4 : 0x87 Brightness Down Hot-Key 5 : Display Scaling Hot-Key 6 : ALS Toggle Hot-Key 15~7 : Reserved
4	LID Event State This value is set after a LID event and cleared after the MXM_FUNC_MXDP sub-function is called. 0 : Not a LID Event (may be Dock, Hot-Plug or other Event) 1 : A LID Event was triggered
5	Dock State This bit reflects the docking state. 0 : Undocked 1 : Docked
7:6	Reserved. Must be zero
13:8	Toggle List Sequence Number The current or expected sequential position of the display combination in the Display toggle list (as described by MDTL) beginning with '1' A '0' value indicates a display combination not expressed in the list, or that the SBIOS has no recommended selection. The method field is evaluated in response to a Display Output Switch Hot-Key Event 0x80, to discover the next desired display combination. A query at any other time returns the current Sequence in the Toggle List.
19:14	Reserved. Must be zero
20	Display Hot-Plug Status 0 : Detached (or LID Closed) 1 : Attached (or LID Open)
28:21	Display Hot-Plug Event The index of the display, as enumerated under _DOD, for which the Hot-Plug Event occurred. For example 1 = 1st display enumerated under _DOD. Zero indicates this is not a Display Hot-Plug
31:29	Reserved. Must be zero

4.3.7.3. MXM_FUNC_MDTL – Display Toggle List

Sub-function 6 returns the hotkey display switch toggle sequence. This function and MXM_FUNC_MXDP are required if the system BIOS expects to exercise control over the detection or selection of displays, such as for notebook display switch hotkeys.

This method is evaluated during early driver initialization and not used again. The toggle list is expressed by the systems as an array of the display IDs, 32-bit display IDs must used. Each combination is separated by a "," symbol 0x2C.

MXM v 3.0 modules are generally assumed to support up to two simultaneous displays. Display combinations which are not available due to a lack of display connectors, module limitations (such as a resource conflict between the listed display outputs) or a lack of connected devices are skipped. If such an entry is skipped the next combination is attempted until one is found which works. After the last display combination, the sequence wraps around starting again at the beginning of the list.

Arguments

Reserved. Must be zero.

On Return

Package containing the Display Toggle List

Example

```
// Function 6: MXM_FUNC_MDTL
case (6) {
    Return ( Package() {
        0x00000110, 0x2C,  // LCD
        0x80000100, 0x2C,  // CRT
        0x80007330, 0x2C,  // HDMI
        0x00000110, 0x80000100, 0x2C, // LCD + CRT
        0x00000110, 0x80007330, 0x2C, // LCD + HDMI
        0x80000100, 0x80007330, 0x2C  // CRT + HDMI
    // roll-over to beginning
    })
}
```


4.3.7.4. MXM_FUNC_MXCB – Query/ Call System Callbacks

Sub-function 25 returns the required system callbacks. The function is called once during driver initialization with null arguments to discover if any system callbacks are needed. It is later called at run-time with the appropriate argument bits set to indicate the callback condition. This function is required only if callbacks from the driver are needed by the SBIOS. If this method is not implemented in the SBIOS, the driver will not generate any of the listed MXM_FUNC_MXCB callbacks.

Arguments

On callback event, the driver indicates the reason for the callback. If a power state transition (bit 2 or bit 3) is one of the reasons, then bits 7:4 are used to fill in the current MXM power state, as enumerated in the Type field in the Input Power Structures (P0 = 0, P1 = 1, Auxiliary Power State 1 = 9, etc.).

Table 4-5. MXM_FUNC_MXCB Callback

Bits	Definition
31:0	Callbacks The bit field indicates the reason for the callback. When set the bit indicates the kind of event causing the Callback. Bit 31~8 : Reserved for future use Bit 7:4 : Current Power State (if bit 2 or bit 3 is set) Bit 3 : Pre power state transition Bit 2 : Post power state transition Bit 1 : Pre Mode Set. Bit 0 : Post Mode-Set All bits = '0' indicates a call to request a list of required callbacks.

On Return

If the input argument is 0, the return buffer contains the requested callbacks the system expects from the GPU driver. This call is typically made once at initialization time.

At runtime, if bit 2 or bit 3 is set in the argument, the SBIOS should return its desired input power state restrictions in bits 7:4, if any. The value provided corresponds to the Type field in the relevant Input Power Structure. A value of '0xF' in bits 7:4 indicates the SBIOS has no preference.

Table 4-6. MXM_FUNC_MXCB Return Buffer Callbacks

Bits	Definition
31:0	Callbacks The index of the requested callback. Bit 10 : Driver will call MXM_FUNC_MXDP sub-function (not MXM_FUNC_MXCB), providing Attached and Active Displays, after a display configuration change has completed (for example, via control panel UI) Bit 9 : Driver will call MXM_FUNC_MXDP sub-function (not MXM_FUNC_MXCB), providing Attached and Active Displays, after a display Hot-Plug (Plug or Unplug) event Bit 8 : Reserved. Bit 7:4 : Current system P-state limit (if bit 2 or 3 is set) Bit 3 : Pre power state transition Bit 2 : Post power state transition Bit 1 : Pre Mode Set Bit 0 : Post Mode-Set A '1' indicates a callback is required for the given event

Example

```
// Function 25: MXM_FUNC_MDTL
case (25) {
    Return ( Buffer() {
        0x0000000           // no callbacks requested
    })
}
```

4.3.7.5. MXM_FUNC_EVENTLIST

Sub-function 18 returns a list of Event Notifiers and their meaning. This sub-function is required only if the system BIOS wishes to reassign notifiers from the default values listed in Table 4-7. For example, to avoid conflict with an existing non-MXM use of a notifier. Only the range of custom codes (0xC0~0xFF) are available for these reassignments, as per the *ACPI 3.0 Specification*.

Any event listed below other than those marked 'Fixed' may be reassigned to other notify codes within this range. The Events are described using GUIDs. There is one GUID per purpose. An event can be assigned to another notifier in the 0xC0-0xFF range by listing it in the event list. Events not specified in the list are assumed to retain their default assignments, as listed in the table. If the table is absent, all default purposes will be assumed for any notifiers the SBIOS chooses to use.

It is permitted to have more than one notifier assigned to the same event (in which case any of those notifiers will trigger the event), but a single notifier cannot be assigned to multiple events.

If an event GUID appears in the list, it is assumed to have been moved from its old notifier to the one specified. If listed more than once, all listed notifiers will trigger that event. Entries listed as 'fixed' can be reassigned, but since these are standard ACPI events that may have uses outside the MXM software, this is not recommended.

Note: Events specified as 'fixed' below are defined in the ACPI 3.0 specification and cannot be deallocated.

Multiple P-state events cannot be mapped to the same notifier, since the notifier implicitly communicates the desired level to the GPU software.

Arguments

Reserved. Must be zero.

Return Values

The return package is a list of event notifier values and the GUIDs representing their new intended function. These GUIDs are selected from the table below.

The following pre-defined GUIDs are those to be used in the table returned through the _DSM sub-function MXM_FUNC_EVENTLIST, and in the WMI-ACPI interface, when referring to the listed event notifiers. Entries listed as 'future' are not yet part of the MXM specification but are provided here in order to help standardize compatibility in the future.

Table 4-7. GUID List for Notification Codes

GUID	Purpose	Default Notify Code
E06BDE62-EE75-48F4-A583-B23E69ABF891	Hot-Key, use _DGS, _DCS ... etc	0x80 (Fixed)
3ADEBD0F-0C5F-46ED-AB2E-04962B4FDCBC	Hot-Plug, query _DCS	0x81 (Fixed)
1E519311-3E75-4208-B05E-EBE17E3FF41F	Backlight Increase	0x86 (Fixed)
37F85341-4418-4F24-8533-38FFC7295542	Backlight Decrease	0x87 (Fixed)
93263095-DA5F-46a0-8B1C-DB2F1F7D66AF	Power State P0	0xD1
BE907006-D603-4714-9AE2-9DBA8997A805	Power State P1	0xD2
21D355E1-498D-4c1f-AE01-05CEF8DD053F	Power State P2	0xD3
EDBCEDF1-BBC9-420e-A3C0-322B1D37C3AA	Power State P3	0xD4
377E84E1-13DE-41fb-A096-0F08528BA689	Power State P4	0xD5
42848006-8886-490E-8C72-2BDCA93A8A09	*Hot-Plug (for Lid State only)*	*0xDB (future)*
44EDE80F-C7E7-4c1c-82AB-9A80050253EB	*Display Scaling change*	*0xDC (future)*
B7175ED9-C995-4d93-8F71-EA25B3A47562	*Ambient Light Sensor*	*0xDD (future)*
B3E485D2-3CC1-4b94-8F31-77BA2FDC9EBE	Update MXM_FUNC_MCPP	0xEF
360D6FB6-1D4E-4fa6-B848-1BE33DD8EC7B	Update MXM_FUNC_MXDP	0xF0

Example

```
//
// Function: MXM_FUNC_EVENTLIST
case (18)
{
    Return ( Package() {
```

```
            0xF0, UUID("93263095-DA5F-46a0-8B1C-DB2F1F7D66AF"),      // Use 0xF0 for P0
            instead of for MXDP update
            0xDF, UUID("360D6FB6-1D4E-4fa6-B848-1BE33DD8EC7B")   // New notifier
            for MXDP update
        })
    }
```

4.3.8. MXDS – Select Display Output Channel

The MXDS method is located inside the namespace of one or more display devices and is used to control the selection of the **Display Output Channel**. It is required if the platform provides MUXes for display output resources which are placed under MXM software control (as opposed to under MXM GPIO control, or no MUXes at all).

If no MXMX method is also present then this one MUX control is assumed to control both display output and DDC. However, separate control of DDC is required in most designs in order to permit detect of all displays without interrupting currently active display outputs. If placed under the parent **Graphics Controller Device** then all MUXed displays under the device shall be switched to that graphics controller when evaluated.

Arguments

Arg0: Get/Set Display State

0 – Get MUX State

1 – Set Display to active on this Output

On Return

If Arg0 = 0, Error = Display is not MUXed

Else return value is the state of the MUX

Example

```
Scope(DGPU_SCOPE) {    // Discrete GPU's display device
    Device(DVI0, 0)}
        Name(ADR, DOD_DVI0)              // DVI on dock
        Method(MXDS, 1) {
            If (And(Arg0, 0)) {
                Return(LNot(\HLMX))    // Sets the MUX toward
            }
            ElseIf (And(Arg0, 1)) {
                Store(Zero, \HLMX)
            }
        }
    }
}
```

4.3.9. MXMX – Select Display Data Channel

This method is required on platforms that MUX DDC/Aux lanes and use a software controlled (as opposed to MXM GPIO controlled) MUX, but is not required on other platforms.

Some designs may share a single display data channel between different display outputs or share access to a single display connector between two GPUs. The MXMX method can be present either in the namespace of a specific display, indicating control for that connector only; or it can be present in the namespace of the parent graphics controller device, indicating control for all signals associated with that GPU.

This method abstracts the platform-specific mechanisms necessary for a client to select the correct signal lines multiplexer. This method is intended for the DDC/Aux channel MUX only. Display output should be controlled through the separate MXDS method.

Additionally, in such designs these data lines may be a shared resource, and a mutex may be internally used to arbitrate access. Callers shall check the return value from an acquire request before assuming they have obtained control. If the caller fails to acquire control they should retry after a reasonable interval (for example: 1~10 mS for 100 Kbps I2C). In all cases, callers shall use this method to release control immediately after multiplexed signal lines are no longer needed.

Arguments

Arg0: Acquire/Release control of multiplexed signal lines

0 – Acquire control

1 – Release control

2 – Get current status

On Return

0, Not Acquired.

If Arg0 = 0 or 1, Non-Zero return indicates success acquiring MUX (and MUX has switched to this output)

If Arg0 = 2, Non-Zero return indicates MUX is currently set to this output

Example

```
Scope(DGPU_SCOPE) {    // Discrete GPU's display device
    Device(DVI0, 0)}
        Name(ADR, DOD_DVI0)           // DVI on dock
```

```
            Method (MXMX, 1, Serialized) {
                Return (0x1)  // No mutex needed, always
                              // always returns success
            }
        }
    }
```

4.3.10. Use of _DOD

In order to identify the available display outputs in a notebook, or docking station, in an interoperable manner, the new ACPI 3.0 format shall be used for enumerating display IDs. This requires all display IDs have set bit [31] and use bits [15:0] to indicate the display type, except for ID 0x110 which is assumed to be the integrated LCD, and only necessary for backwards compatibility.

A display connector with more than one output type may have more than one _DOD value. For example, a DVI-I connector or dual-mode DisplayPort/TMDS.

Note:	The type, topology connection and meaning of the enumerated outputs shall be consistent with other interfaces. For example, what is enumerated in system interfaces such as _DOD shall match devices enumerated from the MXM output device structures. The relative order and number of entries must also be preserved, where applicable. If there are displays which are shared between GPUs, the Display ID will be the same for both GPUs. In the case of a display shared between an MXM and non-MXM GPU, the MXM enumeration must be used.

Refer to the *ACPI Specification* 3.0 for a detailed description of the _DOD method and fields.

Table 4-8. MXM Specific Fields in _DOD

Bits	Definition	
15:0	Device ID. The device ID must be unique under VGA namespace.	
	Bit 3:0	Display Index. This zero-based index provides disambiguation if bits 15:4 are identical for two different entries. If this occurs, increment this index for one of the entries.
	Bit 7:4	Display Port Attachment. This identifies the physical resource used, in this case the pins on the MXM connector. Assigned values are: 0 – analog output (CRT or TV) 1 – LVDS output (LVDS, single or dual-link; TMDS on LVDS output, single or dual-link) 2 – DP_A output (DP, single or dual-link TMDS on DP_A, DP_A + DP_B, DP_A + DP_C) 3 – DP_B output (DP or single-link TMDS on DP_B) 4 – DP_C output (DP, single or dual-link TMDS on DP_C, DP_C + DP_D) 5 – DP_D output (DP or single-link TMDS on DP_D) 6~15 – Reserved. Dual link TMDS using two DP links should use the lower numbered link as its identifier.
	Bit 11:8	Display Type 0 – Other 1 – VGA CRT or VESA Compatible Analog Monitor 2 – TV/HDTV or other Analog Video Monitor 3 – External Digital Monitor 4 – Integrated Flat Panel 5~15 – Reserved for future use
	Bit 15:12	Device Sub-Type. See Below
16	0 – Default 1 – System BIOS has a custom (non-GPU) detect mechanism for the device.	
17	0 – Default 1 – Non-VGA output device whose power is related to the VGA device.	
20:18	The Pipe, Head or GPU	
30:21	0. Reserved for future use	
31	0 – ACPI 2.0 and earlier Legacy IDs. Graphics vendor specific 1 – Uses the ACPI 3.0 bit-field definitions above for bits 15:0	

Notes

Use _DOD to pass extended and backwards compatible legacy IDs, for example:

```
Method(_DOD, 0) {
    Return (Package(){
        0x00000110,// Backward compatible LCD
        0x80000100,// Default CRT
        0x80000210,// Default Composite TV
        0x80000320,// Single Link DVI
        0x80007330,// HDMI Type A
        0x80006340 // DisplayPort 1.1
    })
}
```

1. For Windows XP O/S, use the ID of 0x110 for compatibility with O/S LCD device. Use the 3.0 Extended IDs for all other Devices
2. _DOD requires all connectors to be enumerated, present or not present (e.g. behind a dock even when undocked)
3. Ensure the _ADR is correctly read on all O/Ses the lower 16 bits of _DOD ID must be unique.

4.3.10.1. Type 1 – VGA CRT

Table 4-9. Type 1 – VGA CRT

Bits 15:12	Definition
0	Automatic. Default is VGA 15-pin D-sub
1	DVI-A
2	Apple Mini-VGA
3~15	Reserved for future use

4.3.10.2. Type 2 – Analog TV/ HDTV Connector

Table 4-10. Type 2 - Analog TV/HDTV Connector

Bits 15:12	Definition
0	Automatic. Default is Composite (re-uses Y pin)
1	Composite (re-uses CVBS pin)
2	YprPb Component
3	RGB Component
4	S-video (4 or 7-pin, uses C and Y)
5	SCART with Composite
6	SCART with Composite & RGB
7	SCART with Composite & S-video
8	SMPTE253 Component RGB
9	Japanese D-Connector
9~15	Reserved for future use

4.3.10.3. Type 3 – External Digital Connectors

Table 4-11. Type 3 – External Digital Connectors

Bits 15:12	Definition
0	Automatic. Default is single-link DVI-D
1	DVI-D single-link
2	DVI-D dual-link
3	DVI-I single-link
4	DVI-I dual-link
5	UDI
6	DisplayPort 1.1
7	HDMI 1.2 or 1.3 Type A
8	HDMI 1.2 or 1.3 Type B
9	DisplayPort 1.2
10~15	Reserved for future use

4.3.10.4. Type 4 – Internal Flat Panels

Table 4-12. Type 4 – Internal Flat Panels

Bits 15:12	Definition
0	Automatic. Default is single-link 18-bit LVDS
1	UDI
2	DisplayPort 1.1
3	MIPI DSI
4	HDMI 1.2 Type A
5	HDMI 1.2 Type B
6	Single-link 18-bit LVDS
7	Dual-link 18-bit LVDS
8	Single-link 24-bit LVDS
9	Dual-link 24-bit LVDS
10	eDP (embedded DisplayPort)
11~15	Reserved for future use

4.4. Serial ROM to Access the MXM v 3.0 Structure

A serial ROM on the motherboard is an optional alternative to making the MXM Information Structure available through system methods (Int15h/EFI and ACPI).

The driver and VBIOS will attempt to locate and use the serial ROM before locating the system methods. If both the ROM and system methods are present the structure information will be used from the ROM rather than from system methods.

Communication is through serial link LVDS_DDC and the ROM device is assumed to be at address A8h/A9h. The MXM v 3.0 structure is read out of the ROM by the VBIOS during POST, as a series of sequential bytes starting at offset 0 in the ROM.

Since the link is shared with LVDS_DDC (and, potentially, other internal I2C devices), the use of a serial ROM is not compatible with use of an external display on LVDS_DDC (such as an external DVI or HDMI panel). External displays may interfere with proper I2C operation for other bus devices.

Other system methods (if used) are still required. For example, while the actual displays are defined in a ROM, the system provided methods for display output and DDC lane steering will still be used.

4.4.1. Accessing MXM ROM Via WMI

As the ROM is provided by the system to the module, and connected on a per-module basis, accessing all possible ROM through the primary VGA display device is not viable. For example, if the secondary device is an MXM card from a different vendor, or does not have VBIOS enabled.

If a ROM is present the module can support returning the ROM data to user-mode applications on a per-instance basis, using the WMI format:

```
// WMI MOF Declaration
[
    WMI,
    Dynamic,
    Provider("WMIProv"),
    Locale("MS\\0x409"),
    GUID("{{FEC3A638-6A9F-43f7-BD5A-E1BA4D011E84}}")
]

class MXM30ROMdata
{
    [key, read] String InstanceName;
    [read] Boolean Active;

    [read, WmiDataId(1), Description("MXM v 3.0 sizeof ROM")
    ] uint32 RomSize;

    [read, WmiDataId(2), Description("MXM v 3.0 ROM bytes")
    ] uint8 RomBytes[];
};
```

See Section *Accessing MXM ACPI Methods via WMI* for more details on WMI support in MXM.

5 MXM Structure Field Definitions

This section defines the bit level contents of the MXM system information structure. Parsing of the structure should be required only by graphics module software and the tools used to generate the information structure.

5.1. MXM Header Structure

Table 5-1. MXM Header Structure

Offset	Field Definition	Description
0000 – 0003	"MXM_"	"MXM" header string
0004	Version = 0x03	MXM v 3.0 structure version number. Hex value
0005	Revision = 0x00	MXM v 3.0 structure revision number. Hex value
0006 - 0007	MXM structure length	Byte length of MXM structure including the checksum but not including the MXM header

Note: In the case where multiple structures are embedded in a ROM, the lowest version structure is located first, followed by higher version numbers, in increasing order. The next structure will be located immediately after the last byte of the current structure.

This does not apply to structures returned via system methods. System methods returns only a single structure based on a caller supplied version number.

5.2. MXM Output Device Structure

The DESCRIPTOR field defines what kind of structure information is being stored. A descriptor of 0x00 means this is an output device structure. Each device entry is a 64-bit field, which defines the device type, connection and other parameters of the device.

In general, each physical display connector on the system will have one output device structure. However, dual-mode connectors (DVI-I, or DisplayPort which supports TMDS via a dongle) will have one entry per output mode.

DVI-I uses both the VGA output and one of the TMDS outputs (TMDS over LVDS or through the DisplayPort links). In this case the DDC line specified for the digital portion of the connection is the one that should be used with that connection.

Note: Field Bits [27:23] are overloaded and have different meanings depending on whether an analog TV or digital display is being enumerated.

The type, topology connection and meaning of the enumerated outputs shall match what is enumerated through other system interfaces including ACPI_DOD, Int15h/EFI….etc.

Table 5-2. MXM Output Device Structure

Field Definition	Description
Descriptor[03:00]	Descriptor Type 0x00. Output Device Structure
Device Type[07:04]	Device Type (output resource type on MXM module) 0x00 – Analog CRT 0x01 – Analog TV/HDTV 0x02 – TMDS or HDMI 0x03 – LVDS 0x06 – DisplayPort All other values reserved for future use
DDC/Aux Port [11:08]	DDC or Aux Port connection[1] 0x00 – VGA_DDC 0x01 – LVDS_DDC (also used for external DVI/HDMI over LVDS) 0x09 – Aux[2] Port for DP_A 0x0A – Aux Port for DP_B

[1] In the *MXM Graphics Module Electromechanical Specification Version 3.0*, the DDC port and hotplug pin (if applicable) on the MXM connector which is associated with each device is pre-determined by the device type. As such the value in this field is not selectable, it is pre-determined by the choice of output device.

[2] The Aux pins are shared for both DisplayPort Aux as well as Legacy DDC (dual mode), hence it is not necessary to separately enumerate DDC pins.

Field Definition	Description
	0x0B – Aux Port for DP_C 0x0C – Aux Port for DP_D 0x0F – Not applicable All other values reserved for future use
Connector Type [16:12]	Connector type (output type at system display connector) 0x00 – VGA 0x01 – LVDS 0x02 – HDMI 0x03 – DVI-D 0x04 – DVI-I Analog port 0x05 – DVI-I Digital port 0x06 – DisplayPort external[3] connector 0x07 – DisplayPort internal connector 0x08 – Composite connector on TV_CVBS 0x09 – Composite connector on TV_Y 0x0A – S-video connector on TV_C and TV_Y 0x0B – HDTV connector on HDTV_Y, HDTV_Pr, HDTV_Pb 0x0C – Reserved 0x0D – HDTV connector on HDTV_R, HDTV_G, HDTV_B 0x0E – eDP (embedded DisplayPort) internal connector 0x1F – Not applicable All other values reserved for future use
Connector Location [18:17]	The display output is located: 0x00 – Internal connection, not a user accessible connector 0x01 – A connector integrated in the chassis 0x02 – A connector on a docking station 0x03 – A connector (internal or external) integrated in the chassis which is not available when docked All other values reserved for future use
Digital Connection [22:19] (for digital connection only)	Digital signal MXM pin connection 0x01 – Single Link TMDS (over LVDS – use first link only) 0x02 – Dual Link TMDS (over DisplayPort DP_A+ DP_B) 0x03 – Dual Link TMDS (over DisplayPort DP_A+ DP_C) 0x04 – Dual Link TMDS (over DisplayPort DP_C+ DP_D) 0x05 – Dual Link TMDS (over LVDS) 0x06 – Single Link LVDS (must use first link) 0x07 – Dual Link LVDS 0x0A – DisplayPort DP_A 0x0B – DisplayPort DP_B 0x0C – DisplayPort DP_C 0x0D – DisplayPort DP_D 0x0F – Not applicable All other values reserved for future use

[3] Refer to the DisplayPort specification. The DisplayPort internal connectors are typically used for chassis-internal panels, while the DisplayPort External connector is typically for user accessible external displays.

Field Definition	Description
TV Format [27:23] (for analog TV only)	Default format of TV output for Analog TV/HDTV 0x00 = NTSC_M (US) 0x01 = NTSC_J (Japan) 0x02 = PAL_M (Brazilian format) 0x03 = PAL_BDGHI 0x04 = PAL_N (Paraguay and Uruguay format) 0x05 = PAL_NC (Argentina format) 0x06 = SECAM_L (France) 0x07 = Reserved for future use 0x08 = HD576i 0x09 = HD480i 0x0A = HD480p 0x0B = HD576p 0x0C = HD720p 0x0D = HD1080i 0x0E = HD1080p 0x0F = Not specified. Determined at run time 0x1F = Not applicable All other values reserved for future use
Digital Audio Connection [24:23] (for digital connection only)	Audio for HDMI and DisplayPort 0x00 – Audio over SPDIF connection[4] 0x01 – High definition audio connection (HDA) 0x02 – Audio over PCIe bus 0x03 – Default. No audio connection, or not applicable
Digital Connection Spread Spectrum [25] (for digital connection only)	Enable spread spectrum for digital output 0x00 – Spread spectrum disabled 0x01 – Spread spectrum enabled (default)
Digital CEC [26] (for digital connection ony)	0x00 – Provides CEC signal 0x01 – Default. No CEC
Default Digital LVDS Width [27] (for digital connection only)[5]	0x00 – 24-bit link 0x01 – Default. 18 bit link
GPIO for Output select [32:28]	Optional, set to 0x1F if unused. Specifies the logical GPIO used to select Device Output operation
Polarity for GPIO Output	Specifies the polarity of the GPIO required to select the

[4] The standard MXM v 3.0 connector definitions do not include pins for SPDIF or HDA audio options. If these audio options are supported on an OEM module then OEM pins assigned for these functions must be used.

[5] This bit field sets the default link width for the connection. The LVDS link width can be specified per-panel in its EDID (using EDID v1.4).

Field Definition	Description
Select [33]	Device Output. Note: “Logical” values imply the asserted state as set by software 0 – A logical ‘0’ on the GPIO selects the Output 1 – A logical ‘1’ on the GPIO selects the Output
System Output Method [34]	System Methods (Int15h/EFI and ACPI _DSM) to select display output 0 – Use GPIOs to select Output as per Bits [32:28] 1 – Int15h/EFI and /or ACPI methods can select Output
GPIO for DDC select [39:35]	Optional, set to 0x1F if unused, Specifies the logical GPIO as identified in the used to select DDC for device Note that a logical ‘1’ on the indicated GPIO will steer a DDC MUX to select this Output Device’s DDC lanes.
System DDC Method [40]	System Methods (Int15h/EFI and ACPI MXMX) to select DDC 0 – Use GPIOs to select DDC as per Bits [39:35] 1 – Int15 and/or ACPI methods can select DDC
GPIO for Device Detection [45:41]	Optional, set to 0x1F if unused. Specifies the logical GPIO used to select Device Detection.
Polarity for GPIO Device Detection [46]	Specifies the polarity of the GPIO which indicates device presence. Note: “Logical” values imply the asserted state as set by software. 0 – A logical ‘0’ on the GPIO indicates device is present 1 – A logical ‘1’ on the GPIO indicates device is present
System Hot Plug Notify [47]	The display can cause ACPI Hot Plug Notification (for example, this display can be detected by the system without GPU software involvement – see MXM_FUNC_MXDP). This bit should be set for laptop displays affected by lid state. 0 – No notification 1 – Uses ACPI Notify for Display Plug/Unplug event
Reserved [52:48]	Reserved. Must be zero
LVDS type [55:53] (for digital LVDS connection only)	Signal type for LVDS panel 0 – SPWG 1 – Open LDI 2~7 – Reserved for future use.
Reserved [63:56]	Reserved. Must be zero.

5.3. MXM System Cooling Capability Structure

The **DESCRIPTOR** field defines what kind of structure information is being stored. A descriptor of 0x01 means this is a thermal design power structure. Each cooling capability entry is a 32-bit field.

Table 5-3. MXM System Cooling Capability Structure

Field Definition	Description
Descriptor [03:00]	Descriptor Type 0x01. System Cooling Capability Structure
Type [07:04]	Type of cooling capability information 0x00 – Maximum cooling capability available for the entire module
Value [19:08]	Primary Cooling capability in 0.1 W (100 mW) Example, a value of 0x78 (120) is 12.0 watts and a value of 0x145(325) is 32.5 watts.
Reserved [31:20]	Reserved. Must be zero.

5.4. MXM Thermal Structure

The **DESCRIPTOR** field defines what kind of structure information is being stored. A descriptor of 0x02 means this is a thermal structure. Each thermal entry is a 32-bit field.

Table 5-4. MXM Thermal Structure

Field Definition	Description
Descriptor [03:00]	Descriptor Type 0x02. Thermal Structure
Type [07:04]	Type of thermal information 0x00 – Maximum temperature for module to maintain 0x01 – Temperature to assert MXM TH_ALERT signal 0x02 ~ 0x0F – Reserved for future use
Value [18:08]	Temperature in 0.1 degrees Celsius (0.1 C) Example, a value of 0x3E8 (1000) is 100.0 degrees Celsius.
Reserved [31:19]	Reserved. Must be zero.

5.5. MXM Input Power Structure

The DESCRIPTOR field defines what kind of structure information is being stored. A descriptor of 0x03 means this is an MXM input power structure. Each input power entry is a 32-bit field.

The TYPE field defines the power source level. There will be a separate input power structure for each available power source type or capability level. The TYPE field is 4 bits wide.

The MXM software selects between the available levels on the basis of ACPI notifications from the SBIOS and system input via the PWR_LEVEL# pin. The specifics are provided in section 2.5.

All systems must provide at least one input power structure, of Type 1.

All systems where available module power can be reduced without prior warning, such as the unplugging of a notebook which cannot provide full power to the graphics module while on battery power, must support the PWR_LEVEL# pin and specify in an entry with an input power level Type 0 what power is available in this mode. For the module, fully asserting the module's PWR_LEVEL# pin must cause it to restrict power consumption to be no higher than level.

Only the entry of Type 0 may have a hardware notification value of 1.

Any auxiliary power states defined can be selected using **MXM Auxiliary Power State** notifications (values 0xD1 (no auxiliary power state) - 0xD5).

If no ACPI notifications are available in the system, the ACPI notification for all entries must be set to '1'.

The VALUE field specifies the upper limit on power supplied by the input power type based on the system and module connector capabilities. Power (in Watts) is specified in 0.1 W increments.

Table 5-5. MXM Input Power Structure

Field Definition	Description
Descriptor [03:00]	Descriptor Type 0x03. Input Power Structure
Type [07:04]	Input Power Level 0x00 – power limit when PWR_LEVEL# asserted (logical low) (for example, notebook battery power) 0x01 – default available power when PWR_LEVEL# deasserted (logical high) (for example, notebook AC power) 0x09 ~ 0x0C – Auxiliary Power States P1 ~ P4 All other values reserved for future use.
Hardware Notification [08]	Set to 0 for all entries except Type 0, which may optionally set it to 1 depending on system usage of the PWR_LEVEL# pin. 0 – PWR_LEVEL# pin will not stay asserted during this input power state 1 – PWR_LEVEL# pin will stay asserted during this input power state (this value can only be used for Input Power Structure Type 0

Field Definition	Description
Software Notification [9]	System uses a software mechanism (such as ACPI power source notification or MXM Auxiliary Power State notifiers) to select power levels. (This field should be set to the same value across all Input Power Structure entries.) 0 – System provides ACPI or other software notification to select power states (default) 1 – System provides no software notification to select power states
Reserved [15:10]	Reserved. Must be zero.
Value [27:16]	Value of input in 0.1 W (100 mW) Example, a value of 0x78 (120) is 12.0 watts and a value of 0x145(325) is 32.5 watts.
Reserved [31:28]	Reserved. Must be zero.

5.6. MXM GPIO Device Structure

The **DESCRIPTOR** field defines what kind of structure information is being stored. A descriptor of 0x04 means this is a GPIO device structure. Each GPIO device entry is a 32-bit field, while each GPIO pin is a 16-bit field.

Number GPIO pin entries field defines the number of GPIO pin entry structures which are defined.

Table 5-6. MXM GPIO Device Structure

Field Definition	Description
Descriptor [03:00]	Descriptor Type 0x04. MXM GPIO device structure
Type [11:04]	GPIO Type 0xFF – Direct GPIO (defined on MXM connector)
Reserved [19:12]	Reserved. Must be zero.
Number GPIO pin entries [24:20]	Number of GPIO pin entries.
Reserved [31:25]	Reserved. Must be zero.

Each GPIO pin entry has the following 16-bit structure (Table 5-7):

Table 5-7. GPIO Pin Entry Structure

Name	Description
Logical GPIO Number [04:00]	Logical GPIO number associated to this GPIO pin
Reserved [07:05]	Reserved. Must be zero.
Function [15:08]	This identifies the function of the GPIO pin. 00 = Undefined 01 = Output Device DDC/Aux Bus MUX* 02 = Output Device Display Signal MUX* 03 = Auxiliary Display Detect** 31 = LCD Self Test 32 = LCD Lamp Status 36 = HDTV Select: Allows steering the lines driven between SDTV (Logical '0') and HDTV (Logical '1') 37 = HDTV Alt-Detect: Allows detection of the connectors that are not steered by HDTV Select. That is, if HDTV Select is currently steered towards SDTV, then this GPIO would allow us detect the presence of the HDTV connection. Other function types are reserved.

* The MXM Output Device Structure is used to indicate which output device these GPIOs are associated with, and the polarity.

** There is already one dedicated hotplug pin associated with each suitable display output from the MXM module. This GPIO function should only be used for special situations where an additional display detect pin is needed, such as where an MXM output is shared via a MUX between more than one connector. The MXM Output Device Structure is used to indicate which output device this GPIO is associated with, and the polarity.

Refer to the *MXM version 3.0 Graphics Module Thermal Electromechanical Specification* for the GPIO function hardware definitions.

Each of these GPIO usage models implies an electrical signaling requirement.

Table 5-8. GPIO Pin Usage Methods

Usage Model	Type	Signaling
DDC MUX (combine GPIO Type & Output Display Struct)	Output	Open-Drain
Output MUX (combine GPIO Type & Output Display Struct)	Output	Push-Pull
Display Detect (combine GPIO Type & Output Display Struct)	Input	N/A
D connector line	Output	3-State
D connector plug insertion detect	Input	N/A
LCD Self Test	Output	Push-Pull
LCD Lamp Status	Input	N/A
HDTV Select	Output	Push-Pull
HDTV Alt-Detect	Input	N/A

Following is an example of how some external GPIO chips can implement these signal types. Support for this or for comparable on-GPU capabilities is implemented by the graphics vendor in their MXM software.

Push-Pull : Normal GPIO Output mode

Logical '0' = GPIO in Output Mode & output set to '0'

Logical '1' = GPIO in Output Mode & output set to '1'

3-State : Supports three output states using Output and Input Mode

Logical '00' = GPIO in Output Mode & output set to '0'

Logical '01' = GPIO in Output Mode & output set to '1'

Other = GPIO Input Mode for Hi-Z

Open-Drain : Set output to Low and toggle Input/Output Mode

Logical '0' = GPIO in Output Mode & output set to '0'

Logical '1' = GPIO in Input Mode

Refer to the *MXM Version 3.0 Graphics Module Thermal Electromechanical Specification* for the GPIO function hardware definitions.

5.7. MXM Vendor Specific Structure

The **DESCRIPTOR** field defines what kind of structure information is being stored. A descriptor of 0x05 means this is a vendor specific structure. Each vendor structure entry is a 64-bit field.

The TYPE field indicates the vendor using the 16-bit Vendor Identifier as defined by the Plug & Play specification.

Table 5-9. MXM Vendor Specific Structure

Field Definition	Description
Descriptor [03:00]	Descriptor Type 0x05. GPU vendor specific structure
Type [19:04]	VID – Plug & Play. GPU vendor identifier
Reserved [63:20]	GPU vendor specific contents.

5.8. MXM Backlight Control Structure

The **DESCRIPTOR** field defines what kind of structure information is being stored. A descriptor of 0x06 means this is a backlight control structure. Typically only a single internal display supports direct backlight control by the GPU. Each backlight entry is a 32-bit field. The structure provides hardware information about the backlight on the system. This is used in conjunction with the ACPI-specified list of brightness selections (_BCL) to allow the graphics driver to support brightness control for the backlight.

The Output Device field indicates the position in the output device structure list of the affected display.

The Control Type field indicates the method of backlight control supported in the system.

There can be one or more backlight frequency structures which specify the frequencies and duty cycle ranges which the backlight requires. Viewed together these structures should span the full desired duty cycle range (for example: 100 Hz, 10.0% - 39.9%; 200 Hz, 40.0% - 100.0%).

The MXM specification does not specify a means to dynamically modify brightness until the driver has loaded. The maximum brightness within the first backlight control entry is the brightness that should be used prior to driver load.

The MXM software will map the range of brightness specified in _BCL to duty cycle values between the lowest and highest duty cycles across all backlight frequency entries (most panel backlights support only one frequency, so only one entry is required).

Note: Bits [31:16] are overloaded, with different meaning depending on the Control Type.

Table 5-10. MXM Backlight Control Structure

Field Definition	Description
Descriptor [03:00]	Descriptor Type 0x06. Backlight control structure
Output Device [07:04]	Index of associated Output Device Structure, enumerated in order of occurrence (first Output Device Structure having an index of 0, the second an index of 1, etc.).
Control Type [09:08]	Backlight Inverter Control Type 0 – PWM (from MXM connector pin) 1 – SMBus controller (identified in field) 2 ~ Reserved for future use
Backlight Type [11:10]	Backlight Type 0 – CCFL 1 – LED 2 ~ Reserved for future use
Number of Backlight Frequency Entries [15:12]	Total number of backlight frequency entries (minimum of 1).
Reserved [31:16] (for PWM control type)	Reserved for future use. Must be zero.
SMBus address [23:16] (for SMBus control type)	Backlight controller address on SMBus (if Backlight Inverter Control Type is SMBus).
Controller identifier [31:24] (for SMBus control type)	Controller identifier (for SMBus controllers). 0x00 – TBD (vendor/chip type) 0x01 – TBD (vendor/chip type)

Table 5-11. MXM Backlight Frequency Structure

Field Definition	Description
Duty Cycle Frequency [17:0]	PWM Base Frequency in Hz
Reserved [31:18]	Reserved for future use. Must be zero.
Max Duty Cycle [41:32]	The maximum duty cycle for PWM Inverter at this PWM frequency, in 1/10 %
Min Duty Cycle [51:42]	The minimum duty cycle for PWM Inverter at this PWM frequency, in 1/10 %
Reserved [63:52]	Reserved. Must be zero.

5.9. MXM Fan Control Structure

The **DESCRIPTOR** field defines what kind of structure information is being stored. A descriptor of 0x07 means this is a fan control structure. Each fan control structure has a 64-bit header followed by one or more 32-bit fan speed entries.

The TYPE field indicates the method of fan control supported in the system. The ramp speed specified is the minimum requirement across the full range.

When the module is powered on, the MXM module should assert the fan GPIO (full on) until such time as the MXM software can process the fan structure and set an appropriate speed. By extension, the fan the system provides must be able to support a 100% duty cycle.

Table 5-12. MXM Fan Control Structure

Field Definition	Description
Descriptor [03:00]	Descriptor Type 0x07. MXM fan control structure
Fan Control Type [07:04]	Fan control type 0 – PWM 1 ~ Reserved for future use
Fan Speed Entries [10:08]	Number of fan speed entries following this header (minimum of 1 required).
Reserved [11]	Reserved. Must be zero.
PWM frequency [29:12]	PWM frequency for fan in Hz
Reserved [31:30]	Reserved. Must be zero.
Ramp up [43:32]	Minimum transition time in milliseconds to transition from lowest speed entry to full on.
Ramp down [55:44]	Minimum transition time in milliseconds to transition from full on to lowest speed entry.
Reserved [63:56]	Reserved. Must be zero.

The fan speed structures describe the thermal control behavior the system expects. The temperature field in each entry specifies the lower limit at which the fan speed specified in the entry is to be used. If there is a minimum fan speed (other than full off), it should be specified using an entry with a temperature of 0.

Table 5-13. MXM Fan Speed Structure

Field Definition	Description
Temperature [10:00]	Minimum temperature to enable this entry, in 0.1 degrees Celsius (0.1 C) Example, a value of 0x3E8 (1000) is 100.0 degrees Celsius.
Speed [20:11]	Speed as percentage of full fan range (off to full on) in 1/10%
Reserved [31:21]	Reserved. Must be zero.

Applicable Documents

The following documents contain provisions which through reference in this text, constitute provisions of this standard. At the time of publication, the editions indicated were valid. However, users of this standard are advised to ensure they have the latest versions of referenced standards and documents.

- http://mxm-sig.org - *MXM Version 3.0 Graphics Module Thermal Electromechanical Specification*
- http://www.acpi.info/ - *Advanced Configuration and Power Interface Specification* Revision 3.0a
- http://www.uefi.org/ - *Unified Extensible Firmware Interface Specification*, Version 2.1

Notice

ALL DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE.

Information furnished is believed to be accurate and reliable. However, the developers of this specification assume no responsibility for the consequences of use of such information or for any infringement of patents or other rights of third parties that may result from its use. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all information previously supplied.

Trademarks

PCI Express is a trademark of the PCI-SIG. DisplayPort is a trademark of VESA. HDMI is a trademark of HDMI Licensing LLC. This document refers to specification names that may be trademarks of the respective companies with which they are associated.

Copyright