American Megatrends Inc.
Award Software International Inc.
Dell Computer Corporation
Intel Corporation
Phoenix Technologies Ltd.
SystemSoft Corporation

Desktop Management BIOS Specification

Version 2.0

March 6, 1996

Document Information

The softcopy version of this specification, in Microsoft Word-for-Windows 6.0 format, is available as DMIB20.DOC via ftp://ftp.ptltd.com/pub/phoenix_docs/dmib20.doc or from the Phoenix Technologies World Wide Web site at http://www.ptltd.com/techs/specs.html.

Document Revision History

Version 2.0D	09/14/95	Initial Release of DRAFT COPY
Version 2.0M	12/12/95	Final draft released, with the following changes: - Specified that dmiStorageBase (Function 50h) and NVStorageBase (Function 55h) must be paragraph-aligned. - Added Command value to change a string to function 52h; Command enumeration values modified. - Removed redundant enumerations from Processor Family list - Corrected Memory Subsystem Example - Corrected/clarified Indexed I/O access-methods for event-log; Access Method enumeration values and Access Method Address union modified - Added clarifications to some of the event log types
Version 2.00	03/06/96	Final release, with the following changes: - Specified that all structures end with a terminating NULL, even if the formatted portion of the structure contains string-reference fields and all the string fields are set to 0. - Corrected the Memory Subsystem Example, handles are now correctly created with a 'dw'. - Fixed formatting of some bit definition fields and function examples.

Table Of Contents

1. Overview

Desktop Management Interface (DMI) is a new method of managing computers in an enterprise. The main component of DMI is the Management Information Format Database, or MIF. This database contains all the information about the computing system and its components. Using DMI, a system administrator can obtain the types, capabilities, operational status, installation date, and other information about the system components.

The Desktop Management BIOS Specification documents a standard embedded tool-set to assist in the generation of a system MIF database.

1.1 References

Desktop Management Interface Specification, Version 1.0, April 29, 1994.
DMTF PC Systems Standard MIF Definition, Version 1.3, March 1, 1995.
DMTF Server Standard MIF Definition, Draft Version 0.3, March 1, 1995
Plug and Play BIOS Specification, Version 1.0A, May 5, 1994
PCI BIOS Specification, Version 2.1, August 26, 1994

1.2 Enhancements to the current BIOS architecture

The DMI specification requires that certain information about the System Board be made available to an applications program. For systems implementing DMI BIOS Extensions, user-defined information will be located in a series of data structures. These data structures are accessed by the method described in Section 2.

Vendors may decide to include all or any part of this information in their designs. For a complete solution that is compatible with the Service Layer distributed by the DMTF, vendors must also implement component instrumentation. This instrumentation allows the Service Layer to gain access to the information stored in the BIOS. In addition, a MIF file must be provided that describes that data that is provided by the BIOS and the method of accessing that data. As a minimum, the PC Standard System MIF provided by the DTMF can be used for this purpose.

2. Accessing DMI Information

2.1 Calling Convention

To prevent the proliferation of interfaces for accessing information embedded in the System BIOS, the Desktop Management BIOS Specification will follow the System Device Node model used by Plug and Play, and use Plug and Play BIOS functions to access DMI information. Plug and Play functions 50h-5Fh have been assigned to the DMI BIOS Interface.

Each of the DMI BIOS Plug-and-Play functions is available both in real-mode and 16-bit protected-mode. A function called in 16-bit protected-mode supports both 16-bit and 32-bit stack segments.

2.2 DMI BIOS Functions

This table defines the current DMI BIOS Functions.

DMI BIOS Function	Function Number	Description	Required/Optional
GET_DMI_INFORMATION	50h	Returns the Number of Structures, the Size of the Largest Structure, and the DMI BIOS Revision.	Required
GET_DMI_STRUCTURE	51h	Copies the information for the specified DMI Structure into the buffer specified by the caller.	Required
SET_DMI_STRUCTURE	52h	Copies the information for the specified DMI structure from the buffer specified by the caller.	Optional
GET_DMI_STRUCTURE_ CHANGE_INFO	53h	Returns the DMI Structure Change Information into a 16-byte buffer specified by the caller.	Required for Dynamic Structure-change Notification Support
DMI_CONTROL	54h	Controls a system action	Optional
GET_GPNV_INFORMATION	55h	Returns information about the *G*eneral *P*urpose *N*on-*V*olatile Storage Area	Required for GPNV Support
READ_GPNV_DATA	56h	Reads the entire specified GPNV contents into a buffer specified by the caller.	Required for GPNV Support
WRITE_GPNV_DATA	57h	Copies the contents of the user specified buffer into the GPNV. The function causes the entire specified GPNV to be updated.	Required for GPNV Support
Reserved for Future Use	58h-5Fh	Reserved, will return DMI_FUNCTION_NOT_ SUPPORTED.	Reserved

2.3 Error Return Codes

After the call has been made, the following return codes are available in the AX Register.

Return Code	Value	Description
DMI_SUCCESS	00h	Function Completed Successfully
DMI_UNKNOWN_FUNCTION	81h	Unknown, or invalid, function number passed
DMI_FUNCTION_NOT_SUPPORTED	82h	The function is not supported on this system
DMI_INVALID_HANDLE	83h	DMI Structure number/handle passed is invalid or out of range.
DMI_BAD_PARAMETER	84h	The function detected invalid parameter or, in the case of a "Set DMI Structure" request, detected an invalid value for a to-be-changed structure field.
DMI_INVALID_SUBFUNCTION	85h	The SubFunction parameter supplied on a DMI Control function is not supported by the system BIOS.
DMI_NO_CHANGE	86h	There are no changed DMI structures pending notification.
DMI_ADD_STRUCTURE_FAILED	87h	Returned when there was insufficient storage space to add the desired structure.
DMI_READ_ONLY	8Dh	A "Set DMI Structure" request failed because one or more of the to-be-changed structure fields are read-only.
DMI_LOCK_NOT_SUPPORTED	90h	The GPNV functions do not support locking for the specified GPNV handle.
DMI_CURRENTLY_LOCKED	91h	The GPNV lock request failed - the GPNV is already locked.
DMI_ INVALID_LOCK	92h	The caller has failed to present the predefined *GPNVLock* value which is expected by the BIOS for access of the GPNV area.

2.4 DMI BIOS Structure Access Interface

2.4.1 Function 50h – Get DMI Information

Synopsis:

```
short FAR (*entryPoint)(Function, dmiBIOSRevision, NumStructures, StructureSize, dmiStorageBase,
                        dmiStorageSize, BiosSelector);
short Function;                          /* PnP BIOS Function 50h */
unsigned char FAR *dmiBIOSRevision;      /* Revision of the DMI BIOS Extensions */
unsigned short FAR *NumStructures;       /* Maximum Number of Structures the BIOS will return */
unsigned short FAR *StructureSize;       /* Size of largest DMI Structure */
unsigned long FAR *dmiStorageBase;       /* 32-bit physical base address for memory-mapped */
                                         /* DMI data */
unsigned short FAR *dmiStorageSize;      /* Size of the memory-mapped DMI data */
unsigned short BiosSelector;             /* PnP BIOS readable/writable selector */
```

Description:

Required for DMI BIOS Support. This function will return the revision of the DMI BIOS Extensions and the maximum number of DMI structures that the system BIOS will return information for in *NumStructures*. These structures represent the DMI information that is embedded in the System BIOS. In addition to the number of structures, the system BIOS will return the size, in bytes, of the largest DMI structure (and all of its supporting data) in *StructureSize*. This information can be utilized by the system software to determine the amount of memory required to get all of the DMI structures. *Note*: The system BIOS may return a value that is larger than the actual largest DMI structure to facilitate hot docking or other dynamic DMI information. The BIOS may also return fewer than *NumStructures* when the structures are retrieved using Function 51h. If the BIOS does not support DMI capability, DMI_FUNCTION_NOT_SUPPORTED (82h) will be returned.

The *dmiBIOSRevision* parameter indicates compliance with a revision of this specification. It is a BCD value where the upper nibble indicates the major version and the lower nibble the minor version. For revision 2.0 the returned value will be 20h.

dmiStorageBase is updated by the BIOS call with the paragraph-aligned, 32-bit absolute physical base address of any memory-mapped DMI structure information. If non-zero, this value allows the caller to construct a 16-bit data segment descriptor with a limit of *dmiStorageSize* and read/write access for subsequent input to functions 51h to 54h. If *dmiStorageBase* is 0, protected-mode mapping is not required the DMI structure information and the *dmiStorageSize* return value has no meaning.

The *BiosSelector* parameter enables the system BIOS, if necessary, to update system variables that are contained in the system BIOS memory space. If this function is called from protected mode, the caller must create a data segment descriptor using the 16-bit Protected Mode data segment base address specified in the Plug and Play Installation Check data structure, a limit of 64KB, and the descriptor must be read/write capable. If this function is called from real mode, *BiosSelector* should be set to the Real mode 16-bit data segment address as specified in the Plug and Play Installation Check Structure. Refer to section 4.4 of the Plug and Play BIOS Specification revision 1.0a for more information on the Plug and Play Installation Check Structure and the elements that make up the structure.

This function is available in real mode and 16-bit protected mode.

Returns:

If successful - DMI_SUCCESS
If an Error (Bit 7 set) or a Warning occurred the Error Code will be returned in AX, the FLAGS and all other registers will be preserved.

Example:

The following example illustrates how the 'C' style call interface could be made from an assembly language module:

```
push    BiosSelector
push    segment/selector of dmiStorageSize    ; Pointer to DMIStorageSize
push    offset of dmiStorageSize
push    segment/selector of dmiStorageBase    ; Pointer to DMIStorageBase
push    offset of dmiStorageBase
push    segment/selector of StructureSize     ; Pointer to StructureSize
push    offset of StructureSize
push    segment/selector of NumStructures     ; Pointer to NumStructures
push    offset NumStructures
push    segment/selector of dmiBIOSRevision   ; Pointer to DMIBIOSRevision
push    offset dmiBIOSRevision
push    GET_DMI_INFORMATION                   ; Function number, 50h
call    FAR PTR entryPoint
add     sp, 24                                ; Clean up stack
cmp     ax, DMI_SUCCESS                       ; Function completed successfully?
jne     error
```

2.4.2 Function 51h – Get DMI Structure

Synopsis:

```
short FAR (*entryPoint)(Function, Structure, dmiStrucBuffer, dmiSelector, BiosSelector);
short Function;                    /* PnP BIOS Function 51h */
unsigned short FAR *Structure;     /* Structure number/handle to retrieve*/
unsigned char FAR *dmiStrucBuffer; /* Pointer to buffer to copy structure data to */
unsigned short dmiSelector;        /* DMI data read/write selector */
unsigned short BiosSelector;       /* PnP BIOS readable/writable selector */
```

Description:

Required for DMI BIOS Support. This function will copy the information for the specified DMI Structure into the buffer specified by the caller. The *Structure* argument is a pointer to the unique DMI Structure number (handle). If *Structure* contains zero, the system BIOS will return the first DMI Structure. The *dmiStrucBuffer* argument contains the pointer to the caller's memory buffer. If the function returns either DMI_SUCCESS or DMI_INVALID_HANDLE, *Structure* is updated with either the next sequential structure handle or the end-of-list indicator 0FFFFh.

The protected-mode read/write selector *dmiSelector* has base equal to *dmiStorageBase* and limit of at least *dmiStorageSize* — so long as the *dmiStorageBase* value returned from Function 50h was non-zero.

The *BiosSelector* parameter enables the system BIOS, if necessary, to update system variables that are contained in the system BIOS memory space. If this function is called from protected mode, the caller must create a data segment descriptor using the 16-bit Protected Mode data segment base address specified in the Plug and Play Installation Check data structure, a limit of 64KB, and the descriptor must be read/write capable. If this function is called from real mode, *BiosSelector* should be set to the Real mode 16-bit data segment address as specified in the Plug and Play Installation Check Structure. Refer to section 4.4 of the Plug and Play BIOS Specification revision 1.0a for more information on the Plug and Play Installation Check Structure and the elements that make up the structure.

This function is available in real mode and 16-bit protected mode.

Returns:
If successful - DMI_SUCCESS
If an Error (Bit 7 set) or a Warning occurred, the Error Code will be returned in AX, the FLAGS and all other registers will be preserved

Example:
The following example illustrates how the 'C' style call interface could be made from an assembly language module:

```
push    BiosSelector
push    dmiSelector
push    segment/selector of dmiStrucBuffer; Pointer to dmiStrucBuffer
push    offset of dmiStrucBuffer
push    segment/selector of Structure      ; Pointer to Structure
push    offset of Structure
push    GET_ DMI_STRUCTURE                 ; Function number, 51h
call    FAR PTR entryPoint
add     sp, 14                             ; Clean up stack
cmp     ax, DMI_SUCCESS                    ; Function completed successfully?
jne     error
```

2.4.3 Function 52h – Set DMI Structure

Synopsis:
```
short FAR (*entryPoint)(Function, dmiDataBuffer, dmiWorkBuffer, Control, dmiSelector, BiosSelector)
short Function;                          /* PnP BIOS Function 52h */
unsigned char FAR *dmiDataBuffer;        /* Pointer to buffer containing new/change data */
unsigned char FAR *dmiWorkBuffer;        /* Pointer to work buffer area for the BIOS */
unsigned char Control;                   /* Conditions for  performing operation */
unsigned short dmiSelector;              /* DMI data read/write selector */
unsigned short BiosSelector;             /* PnP BIOS readable/writeable selector */
```

Description:
Optional. This function will set the DMI structure identified by the type (and possibly handle) found in the DMI structure header in the buffer pointed to by *dmiDataBuffer*. Values that the BIOS allows to be set in the supplied structure will either be updated by the call, or will cause the BIOS to perform some defined action (such as enabling a hardware option, etc.).

Unless otherwise specified, all structures and structure values defined in Section 3, *DMI BIOS Structures*, are read-only and cannot be set. Attempts to set these structures will return a DMI_READ_ONLY error. A structure field that is composed of read/write and read-only subfields can still be set -- so long as the read-only portion of the field is unmodified. Attempting to write to a read-only subfield will also cause a DMI_READ_ONLY to be returned.

The *dmiDataBuffer* parameter references a structure of the following format:

Offset	Field	Length	Description
00h	*Command*	BYTE	Identifies the structure-setting operation to be performed, one of: 00h A single byte of information is to be changed in the structure identified by StructureHeader 01h A word (two bytes) of information is to be changed in the structure identified by StructureHeader 02h A double-word (four bytes) of information is to be changed in the structure identified by StructureHeader 03h The structure identified by StructureHeader is to be added to the DMI structure pool 04h The structure identified by StructureHeader is to be deleted from the DMI structure pool 05h A string's value is to be changed in the structure identified by StructureHeader. 06h-0FFh Reserved for future assignment by this specification.
01h	*FieldOffset*	BYTE	For a structure change Command, identifies the starting offset within the changed structure's fixed data of the to-be-changed item. For a string-value change Command, identifies the offset within the structure's fixed data associated with the string's "number". This field is ignored for all other Commands.
02h	*ChangeMask*	DWORD	For a structure-change Command, identifies the ANDing mask to be applied to the existing structure data prior to applying the ChangeValue. The number of significant bytes within this area is defined by the Command. This field is ignored for all other Commands.
06h	*ChangeValue*	DWORD	For a structure-change Command, identifies the data value to be ORed with the existing structure data – after applying the ChangeMask. The number of significant bytes within this area is defined by the Command. This field is ignored for all other Commands.
0Ah	*DataLength*	WORD	For a structure-add Command, identifies the full length of the to-be-added structure. The length includes the structure header, the fixed-length portion of the structure, and any string data which accompanies the added structure – including all null-terminators. For a string-value change Command, identifies the length of the string data (including the null-terminator); if the length is 1, the current string is deleted. This field is ignored for all other Commands.
0Ch	*StructureHeader*	4 BYTEs	Contains the structure header (see Structure Header Format on page 22) of the structure to be added, changed, or deleted.
10h	*StructureData*	Var	For a structure-add Command, contains the data to be associated with the DMI BIOS Structure identified by the StructureHeader. For a string-value change Command, contains the string's data (the number of characters is identified by DataLength). This field is ignored for all other Commands.

The *dmiWorkBuffer* parameter references a work buffer for use by the BIOS in performing the request; the contents of the buffer are destroyed by the BIOS' processing. This work buffer must be read/write and sized to hold the entire DMI structure pool, based on the maximum structure-size information (*StructureSize* * *NumStructures*)

returned by *Function 50h – Get DMI Information* (see page 7) plus the size of any structure to be added by the request.

The *Control* flag provides a mechanism for indicating to the BIOS whether the set request is to take effect immediately, or if this is a check to validate the to-be-updated data.

Control is defined as:

Bit 0	0 = Do not set the specified structure, but validate its parameters. 1 = Set the structure immediately.
Bits 1:7	Reserved, must be 0.

If bit 0 of *Control* is 0, then the *dmiDataBuffer* values are checked for validity. If any are not valid, then the function returns DMI_BAD_PARAMETER; if any read-only field is modified, the function returns DMI_READ_ONLY. Validity checking is useful to determine if the BIOS supports setting a structure field to a particular value – or whether the BIOS supports writing to a specific structure field. For example, it may be useful for an OEM to determine beforehand whether the OEM's BIOS supports a "Reboot to Diagnostics Now" setting in an OEM-defined structure.

The protected-mode read/write selector *dmiSelector* has base equal to *dmiStorageBase* and a limit of at least *dmiStorageSize*, so long as the *dmiStorageBase* returned from *Function 50h – Get DMI Information* was non-zero.

The *BiosSelector* parameter enables the system BIOS, if necessary, to update system variables that are contained in the system BIOS memory space. If this function is called from protected mode, the caller must create a data segment descriptor using the 16-bit Protected Mode data segment base address specified in the Plug and Play Installation Check data structure, a limit of 64KB, and the descriptor must be read/write capable. If this function is called from real mode, BiosSelector should be set to the Real mode 16-bit data segment address as specified in the Plug and Play Installation Check Structure. Refer to section 4.4 of the Plug and Play BIOS Specification revision 1.0a for more information on the Plug and Play Installation Check Structure and the elements that make up the structure.

This function is available in real mode and 16-bit protected mode.

Note: If the system BIOS supports structure-change notification, a structure-change event will be issued by the BIOS upon its successful completion of a structure-setting (rather than validation) function call. See *Structure Change Notification Interface* on page 12 for more information.

Returns:

If successful - DMI_SUCCESS
If an error occurred, the Error Code will be returned in AX. The FLAGS and all other registers will be preserved.

Errors:

DMI_BAD_PARAMETER	A parameter contains an invalid or unsupported value.
DMI_READ_ONLY	A parameter is read-only and differs from the present value – an attempt was made to modify a read-only value.
DMI_ADD_STRUCTURE_FAILED	The desired structure could not be added due to insufficient storage space.
DMI_INVALID_HANDLE	For an add (03h) *Command*, the structure handle present in the *StructureHeader* already exists or, for a change (00h to 02h and 05h) or delete (04h) *Command*, the structure handle does not exist.

Example:
The following example illustrates how the 'C' style call interface could be made from an assembly language module:

```
push    BiosSelector
push    dmiSelector
push    Control
push    segment/selector of dmiWorkBuffer    ;pointer to BIOS temporary buffer
push    offset of dmiWorkBuffer
push    segment/selector of dmiDataBuffer    ; pointer to structure
push    offset of dmiDataBuffer
push    SET_DMI_STRUCTURE                    ; Function number, 52h
call    FAR PTR entryPoint
add     sp, 16                               ; clean stack
cmp     ax, DMI_SUCCESS                      ; Successful?
jne     error                                ; No, go handle error
```

2.5 Structure Change Notification Interface

Certain classes of systems may provide the capability for the addition or removal of system devices while the system unit is powered on, such as inserting a Notebook unit into a Docking Station. System BIOS support is necessary for providing DMI Structure Change Notification accessible to system software so that when devices are added or removed the system software will comprehend any changes in the DMI BIOS Structures. Structure Change Notification can be implemented as either a polled method or as asynchronous Plug-and-Play events. For information on how Plug-and-Play event notification is accessed, see section 4.6 of the Plug and Play BIOS Specification revision 1.0a.

When system software is notified on an event by either mechanism, it can then call the BIOS runtime function (Plug and Play BIOS Function 3 - Get Event) to get the type of event. In addition to the events defined in the Plug and Play BIOS Specification, the following event has been defined.

Note: Some DMI structure values might be inherently changing (e.g. an OEM-specific structure which returns system temperature and voltage values). Due to the frequency of the values' change, the BIOS might not return Structure Change status for this type of structure.

DMI_STRUCTURE_CHANGE_EVENT 7FFFh

This message indicates that there has been a change in the DMI Information being maintained by the System BIOS. Upon receiving a DMI_STRUCTURE_CHANGE_EVENT, system software can call the BIOS runtime function 53h (Get Structure Change Information) to determine the exact cause of the DMI structure-change event.

2.5.1 Function 53h – Get Structure Change Information

Synopsis:

```
short FAR (*entryPoint)(Function, dmiChangeStructure, dmiSelector, BiosSelector);
short Function;                          /* PnP BIOS Function 53h */
unsigned char FAR *dmiChangeStructure;   /* Pointer to DMI Change structure */
unsigned short dmiSelector;              /* DMI data read/write selector */
unsigned short BiosSelector;             /* PnP BIOS readable/writable selector */
```

Description:

Required for DMI BIOS Dynamic Structure Change Notification Support. This function will allow system software to get information about what type of DMI structure-change occurred. The DMI structure-change information will be returned in the 16-byte memory buffer pointed to by *dmiChangeStructure* in the following format:

Field	Offset	Length	Value
DMI Change Status	00h	BYTE	ENUM
DMI Change Type	01h	BYTE	Bit Field
DMI Structure Handle	02h	WORD	Varies
Reserved	04h-0Fh	12 BYTEs	00h

DMI Change Status:

00h	No Change
01h	Other
02h	Unknown
03h	Single DMI Structure Affected
04h	Multiple DMI Structures Affected
05h - 0FFh	Reserved

DMI Change Type:

Bit 0	One or more structures was changed, when 1.
Bit 1	One or more structures was added, when 1. See "*Function 52h – Set DMI Structure*" for information about adding DMI structures.
Byte 2:7	Reserved, must be 0

If DMI Change Status 03h (Single Structure Affected) is returned, the number (or handle) of the affected structure is present in the "DMI Structure Handle" field; DMI Change Type identifies whether the structure was changed (01h) or added (02h).

If DMI Change Status 04h (Multiple DMI Structures Affected) is returned, the caller must enumerate all the structures to determine what was changed and/or added. DMI Change Type identifies whether multiple structures were changed (01h), multiple structures were added (02h), or structures were both changed and added (03h).

The DMI Change Status Byte remains valid until Function 53h is called. The calling of Function 53h will reset the DMI Change Status Byte to be reset to zero. If the call is issued in the absence of a DMI event, the function returns error code 86h (DMI_NO_CHANGE).

The protected-mode read/write selector *dmiSelector* has base equal to *dmiStorageBase* and limit of at least *dmiStorageSize* — so long as the *dmiStorageBase* value returned from Function 50h was non-zero.

The *BiosSelector* parameter enables the system BIOS, if necessary, to update system variables that are contained in the system BIOS memory space. If this function is called from protected mode, the caller must create a data segment descriptor using the 16-bit Protected Mode data segment base address specified in the Plug and Play Installation Check data structure, a limit of 64KB, and the descriptor must be read/write capable. If this function is called from real mode, *BiosSelector* should be set to the Real mode 16-bit data segment address as specified in the Plug and Play Installation Check Structure. Refer to section 4.4 of the Plug and Play BIOS Specification revision 1.0a for more information on the Plug and Play Installation Check Structure and the elements that make up the structure.

This function is available in real mode and 16-bit protected mode.

Returns:

If successful - DMI_SUCCESS
If an Error (Bit 7 set) or a Warning occurred the Error Code will be returned in AX, the FLAGS and all other registers will be preserved

Example:

The following example illustrates how the 'C' style call interface could be made from an assembly language module:

```
push    BiosSelector
push    dmiSelector
push    segment/selector of dmiChangeStructure
push    offset of dmiChangeStructure
push    GET_ DMI_STRUCTURE_CHANGE_INFO      ; Function number, 53h
call    FAR PTR entryPoint
add     sp, 10                              ; Clean up stack
cmp     ax, DMI_SUCCESS                     ; Function completed successfully?
jne     error
```

2.6 Control Interface

2.6.1 Function 54h – DMI Control

Synopsis:

```
short FAR (*entryPoint)(Function, SubFunction, Data, Control, dmiSelector, BiosSelector)
short Function;                 /* PnP BIOS Function 54h */
short SubFunction;              /* Defines the specific control operation */
void FAR *Data;                 /* Input/output data buffer, SubFunction specific */
unsigned char Control;          /* Conditions for setting the structure */
unsigned short dmiSelector;     /* DMI data read/write selector */
unsigned short BiosSelector;    /* PnP BIOS readable/writeable selector */
```

Description:

Optional. This function provides the interface to perform implementation-specific functions for the system, as defined by the *SubFunction* parameter and its (optional) *Data* values.

SubFunction	Name	Description
0000h	DMI_CLEAR_EVENT_LOG	Clears the event log as described in *System Event Log (Type 15)* on page 45. The *Data* parameter is reserved and must be set to 0.
0001h	DMI_CONTROL_LOGGING	*Data* points to a 2-word (4-byte) buffer that describes how to control event logging – see 2.6.2 for bit-wise definitions. The first word (offset 0:1) identifies the ANDing mask to be applied to the existing log-control value prior to ORing the second word (offset 2:3). The second word is modified by the BIOS to contain the log-control value on entry to this function.
0002h-3FFFh	Reserved	Reserved for future definition by this specification.
4000h-7FFFh	Reserved for BIOS vendor	Available for use by the BIOS vendor.
8000h-FFFFh	Reserved for system vendor	Available for use by the system vendor.

Note: A BIOS might support the Log Control function but not support all the *SubFunction* values.

The *Control* flag provides a mechanism for indicating to the BIOS whether the operation is to be performed immediately, or if this is a check to validate the operation's availability and/or data.

Control is defined as:

Bit 0	0 = Do not perform the operation, but validate its parameters. 1 = Perform the operation immediately.
Bits 1:7	Reserved, must be 0.

If bit 0 of *Control* is 0, then the *SubFunction* and contents of *Data* are checked for validity. If any are not valid, then the function returns DMI_BAD_PARAMETER. Validity checking is useful to determine if the BIOS supports a specific DMI Control *SubFunction*.

The protected-mode read/write selector *dmiSelector* has base equal to *dmiStorageBase* and limit of at least *dmiStorageSize* — so long as the *dmiStorageBase* value returned from Function 50h was non-zero.

The *BiosSelector* parameter enables the system BIOS, if necessary, to update system variables that are contained in the system BIOS memory space. If this function is called from protected mode, the caller must create a data

segment descriptor using the 16-bit Protected Mode data segment base address specified in the Plug and Play Installation Check data structure, a limit of 64KB, and the descriptor must be read/write capable. If this function is called from real mode, BiosSelector should be set to the Real mode 16-bit data segment address as specified in the Plug and Play Installation Check Structure. Refer to section 4.4 of the Plug and Play BIOS Specification revision 1.0a for more information on the Plug and Play Installation Check Structure and the elements that make up the structure.

This function is available in real mode and 16-bit protected mode.

Returns:

If successful - DMI_SUCCESS
If an error occurred, the Error Code will be returned in AX. The FLAGS and all other registers will be preserved.

Errors:

DMI_BAD_PARAMETER The *Data* contents were not valid for the requested SubFunction.

DMI_INVALID_SUBFUNCTION The *SubFunction* requested is not supported by the system BIOS.

Example:

The following example illustrates how the 'C' style call interface could be made from an assembly language module:

```
push    BiosSelector
push    dmiSelector
push    Control
push    segment/selector of Data        ; pointer to SubFunction data
push    offset of Data
push    SubFunction
push    DMI_CONTROL                     ; Function number, 54h
call    FAR PTR entryPoint
add     sp, 14                          ; clean stack
cmp     ax, DMI_SUCCESS                 ; Successful?
jne     error                           ; No, go handle error
```

2.6.2 DMI_CONTROL_LOGGING Control Word

Word Bit Position	Meaning if Set
0	Enable Event Logging (overall)
1	Enable Correctable Memory Error Events' Logging
2 - 15	Reserved for future assignment by this specification.

2.7 General Purpose Nonvolatile Storage Interface

A General-Purpose NonVolatile (GPNV) area is a persistent general-purpose storage area managed by the Desktop Management BIOS. Multiple GPNV areas can be supported by a particular BIOS implementation. The size, format and location of a GPNV are not defined by this specification nor is the number of GPNV areas — these attributes are OEM-specific.

A GPNV storage area is not a requirement for a Desktop Management BIOS. It is one method that might be used to store the System Event Log (see section 3.2.16, page 45). A GPNV storage area is not necessarily dedicated to the Desktop Management functions of the BIOS, it can also be used by other services which require non-volatile storage.

A *Handle* parameter is passed into the GPNV function calls to specify which GPNV area is to be accessed. The *Handle* for the first GPNV area is 0, with remaining GPNV areas identified by *Handle* values 1, 2, 3... *n*, where (*n+1)* is the total number of GPNV areas supported by a particular BIOS implementation.

A *GPNVLock* parameter provides a mechanism for cooperative use of the GPNV. The *GPNVLock* value is set on a Read GPNV request (function 56h) and cleared on a Write GPNV request (function 57h). The BIOS compares the value of the *GPNVLock* which is set on a Read GPNV request with the value of the *GPNVLock* passed as a parameter into the GPNV Write request — if they match, the GPNV Write request succeeds and the GPNV data area will be updated on completion of the GPNV Write; if the lock values do not match, the BIOS does not update the GPNV area and DMI_CURRENTLY_LOCKED is returned. *Note*: GPNV locks are held until unlocked, even through system power and reboot cycles. The method used to preserve the GPNV Locks through boot cycles is left up to the system designer.

A BIOS might choose to "hide" a GPNV area by defining a special lock value which is required to access the area. In this case, the special *GPNVLock* value must be supplied with the GPNV read and write requests or the function is failed by the BIOS with DMI_INVALID_LOCK.

A lock set request *succeeds* when there is no outstanding lock set at the time that the Read GPNV request (Function 56h) is made. A lock set request *fails* when there is already a lock set as the result of a previous Read GPNV request (which has not yet been cleared with a Function 57h Write GPNV request) or when a predefined lock value is required in order to access a particular GPNV area and the *GPNVLock* value provided by the caller does not match the required value.

The BIOS makes no attempt to enforce mutually-exclusive access to the GPNV — it is up to callers of GPNV Read to ensure unique *GPNVLock* values (e.g. process ID).

2.7.1 Function 55H – Get General-Purpose NonVolatile Information

Synopsis:

```
short FAR (*entryPoint)(Function, Handle, MinGPNVRWSize, GPNVSize, NVStorageBase,
                BiosSelector);
short Function;                     /* PnP BIOS Function 55h */
unsigned short FAR *Handle;         /* Identifies which GPNV to access */
unsigned short FAR *MinGPNVRWSize;  /* Minimum buffer size in bytes for accessing GPNV */
unsigned short FAR *GPNVSize;       /* Size allocated for GPNV within the R/W Block */
unsigned long FAR *NVStorageBase;   /* 32-bit  physical base address for... */
                                    /* ... mem. mapped nonvolatile storage media */
unsigned short BiosSelector;        /* PnP BIOS readable/writable selector */
```

Description: *Required for GPNV support.* This function returns information about a General Purpose NonVolatile (GPNV) area. The *Handle* argument is a pointer to a number that identifies which GPNV's information is requested, a value of 0 accesses the first (or only) area.

On return:

**Handle* is updated either with the handle of the next GPNV area or, if there are no more areas, 0FFFFh. GPNV handles are assigned sequentially by the system, from 0 to the total number of areas (minus 1).

**MinGPNVRW Size* is updated with the minimum size, in bytes, of any buffer used to access this GPNV area. For a Flash based GPNV area, this would be the size of the Flash block containing the actual GPNV.

**GPNVSize* is updated with the size, in bytes, of this GPNV area (which is less than or equal to the *MinGPNVRWSize* value).

*NVStorageBase is updated with the paragraph-aligned, 32-bit absolute physical base address of this GPNV. If non-zero, this value allows the caller to construct a 16-bit data segment descriptor with a limit of MinGPNVRWSize and read/write access. If the value is 0, protected-mode mapping is not required for this GPNV.

Returns:

If successful - DMI_SUCCESS

If an Error (Bit 7 set) or a Warning occurred the Error Code will be returned in AX, the FLAGS and all other registers will be preserved

Example:

The following example illustrates how the 'C' style call interface could be made from an assembly language module:

```
push    BiosSelector
push    segment/selector of NVStorageBase
push    offset of NVStorageBase
push    segment/selector of GPNVSize
push    offset of GPNVSize
push    segment/selector of MinGPNVRWSize
push    offset of MinGPNVRWSize
push    segment/selector of Handle
push    offset of Handle
push    GET_GPNV_INFORMATION    ; Function number, 55h
call    FAR PTR entryPoint
add     sp, 20                  ; Clean up stack
cmp     ax, DMI_SUCCESS         ; Function completed successfully?
jne     error
```

2.7.2 Function 56H – Read General-Purpose NonVolatile Data

Synopsis:

```
short FAR (*entryPoint)(Function, Handle, GPNVBuffer, GPNVLock, GPNVSelector, BiosSelector);
short Function;                 /* PnP BIOS Function 56h */
unsigned short Handle;          /* Identifies which GPNV is to be read */
unsigned char FAR *GPNVBuffer;  /* Address of buffer in which to return GPNV */
short FAR *GPNVLock;            /* Lock value */
unsigned short GPNVSelector;    /* Selector for GPNV Storage */
unsigned short BiosSelector;    /* PnP BIOS readable/writable selector */
```

Description: *Required for GPNV support.* This function is used to read an entire GPNV area into the buffer specified by *GPNVBuffer*. It is the responsibility of the caller to ensure that *GPNVBuffer* is large enough to store the entire GPNV storage block - this buffer must be at least the *MinGPNVRWSize* returned by Function 55h - Get GPNV Information. The *Handle* argument identifies the specific GPNV to be read. On a successful read of a GPNV area, that GPNV area will be placed in the *GPNVBuffer* beginning at offset 0. The protected-mode selector *GPNVSelector* has base equal to *NVStorageBase* and limit of at least *MinGPNVRWSize* — so long as the *NVStorageBase* value returned from Function 55h was non-zero.

Passing a *GPNVLock* value of -1 to the GPNV Read causes the *GPNVLock* value to be ignored — in this case the underlying logic makes no attempt to store a lock value for comparison with lock values passed into GPNV Write. Any value provided for *GPNVLock* besides -1 is accepted as a valid value for a lock request.

Returns:

If the GPNV lock is supported and the lock set request succeeds, the caller's *GPNVLock* is set to the value of the current lock and the function returns DMI_SUCCESS.

If the GPNV request fails, one of the following values is returned:

- DMI_ LOCK_NOT_SUPPORTED
- DMI_ INVALID_LOCK
- DMI_ CURRENTLY_LOCKED

For return status codes DMI_SUCCESS, DMI_LOCK_NOT_SUPPORTED and DMI_CURRENTLY_LOCKED, the GPNV Read function returns the current contents of the GPNV associated with *Handle* as the first *GPNVSize* bytes within *GPNVBuffer*, starting at offset 0. If a lock request fails with DMI_CURRENTLY_LOCKED status, the caller's GPNVLock will be set to the value of the current lock.

Example:

The following example illustrates how the 'C' style call interface could be made from an assembly language module:

```
push    BiosSelector
push    GPNVSelector
push    segment/selector of  GPNVLock
push    offset of GPNVLock
push    segment/selector of GPNVBuffer
push    offset of GPNVBuffer
push    Handle
push    READ_GPNV_DATA          ; Function number, 56h
call    FAR PTR entryPoint
add     sp, 16                  ; Clean up stack
cmp     ax, DMI_SUCCESS         ; Function completed successfully?
jne     error
```

2.7.3 Function 57H – Write General-Purpose NonVolatile Data

Synopsis:

```
short FAR (*entryPoint)(Function, Handle, GPNVBuffer, GPNVLock, GPNVSelector, BiosSelector);
short Function;                 /* PnP BIOS Function 57h */
unsigned short Handle;          /* Identifies which GPNV is to be written */
unsigned char FAR *GPNVBuffer;  /* Address of buffer containing complete GPNV to write*/
short GPNVLock;                 /* Lock value */
unsigned short GPNVSelector;    /* Selector for GPNV Storage */
unsigned short BiosSelector;    /* PnP BIOS readable/writable selector */
```

Description: *Required for GPNV support.* This function is used to write an entire GPNV from the *GPNVBuffer* into the nonvolatile storage area. The *Handle* argument identifies the specific GPNV to be written. The protected-mode selector *GPNVSelector* has base equal to *NVStorageBase* and limit of at least *MinGPNVRWSize* — so long as the *NVStorageBase* value returned from *Get GPNV Information* was non-zero. The caller should first call *Read GPNV Data* (with a lock) to get the current area contents, modify the data, and pass it into this function — this ensures that the *GPNVBuffer* which is written contains a complete definition for the entire GPNV area. If the BIOS uses some form of block erase device, the caller must also allocate enough buffer space for the BIOS to store all data from the part during the reprogreamming operation, not just the data of interest.

The data to be written to the GPNV selected by *Handle* must reside as the first *GPNVSize* bytes of the *GPNVBuffer*. *Note*: The remaining (*MinGPNVRWSize-GPNVSize)* bytes of the *GPNVBuffer* area are used as a scratch-area by the BIOS call in processing the write request; the contents of that area of the buffer are destroyed by this function call.

The *GPNVLock* provides a mechanism for cooperative use of the GPNV, and is set during a GPNV Read (Function 56h). If the input *GPNVLock* value is -1 the caller requests a forced write to the GPNV area, ignoring any outstanding *GPNVLock*. If the caller is not doing a forced write, the value passed in *GPNVLock* to the GPNV Write must be the same value as that (set and) returned by a previous GPNV Read (Function 56h).

Returns:

The GPNV Write function returns a value of DMI_ LOCK_NOT_SUPPORTED when a *GPNVLock* value other than -1 is specified and locking is not supported. A return status of DMI_ CURRENTLY_LOCKED indicates that the call has failed due to an outstanding lock on the GPNV area which does not match the caller's *GPNVLock* value. Any outstanding *GPNVLock* value (which was set by a previous *Function 56H – Read General-Purpose NonVolatile Data*) gets cleared on a successful write of the GPNV.

Example:

The following example illustrates how the 'C' style call interface could be made from an assembly language module:

```
push    BiosSelector
push    GPNVSelector
push    GPNVLock
push    segment/selector of GPNVBuffer
push    offset of GPNVBuffer
push    Handle
push    WRITE_GPNV_DATA         ; Function number, 57h
call    FAR PTR entryPoint
add     sp, 14                  ; Clean up stack
cmp     ax, DMI_SUCCESS         ; Function completed successfully?
jne     error
```

3. DMI BIOS Structures

The total number of structures can be obtained through the *Get DMI Information* function (see 2.4.1 on page 7). The System Information is presented to an application as a set of structures that are obtained by calling the *Get DMI Structure* function once per structure (see 2.4.2 on page 8).

Note: All numbers are in decimal format unless otherwise indicated.
'h' indicates hexadecimal format.
'b' indicates binary format

3.1 Structure Standards

Each DMI structure has a formatted section and an optional unformed section. The formatted section of each structure begins with a 4-byte header. Remaining data in the formatted section is determined by the structure type, as is the overall length of the formatted section. The unformed section of the structure is used for passing variable data such as text strings, see 3.1.2 Text Strings for more information.

3.1.1 Structure Header Format

Each DMI BIOS structure begins with a 4-byte header, as follows:

Offset	Name	Length	Description
00h	Type	BYTE	Specifies the type of structure. Types 0 through 127 (7Fh) are reserved for and defined by this specification. Types 128 through 256 (80h to FFh) are available for system- and OEM-specific information.
01h	Length	BYTE	Specifies the length of the formatted area of the structure, starting at the *Type* field. The length of the structure's string-set is not included.
02h	Handle	WORD	Specifies the structure's handle, a unique 16-bit number in the range 0 to 0FFFEh. The handle can be used with the *Get DMI Structure* function to retrieve a specific structure; the handle numbers are not required to be contiguous. If the system configuration changes, a previously assigned handle might no longer exist. However once a handle has been assigned by the BIOS, the BIOS cannot not re-assign that handle number to another structure.

3.1.2 Text Strings

Text strings associated with a given DMI structure are returned in the *dmiStrucBuffer*, appended directly after the formatted portion of the structure. This method of returning string information eliminates the need for application software to deal with pointers embedded in the DMI structure. Each string is terminated with a null (00h) BYTE and the set of strings is terminated with an additional null (00h) BYTE. When the formatted portion of a DMI structure references a string, it does so by specifying a non-zero string number within the structure's string-set. For example, if a string field contains 02h, it references the second string following the formatted portion of the DMI structure. If a string field references no string, a null (0) is placed in that string field. If the formatted portion of the structure contains string-reference fields and all the string fields are set to 0 (no string references), the formatted section of the structure is followed by two null (00h) BYTES.

Note: Each text string is limited to 64 significant characters due to system MIF limitations.

Example: BIOS Information

```
BIOS_Info    LABEL BYTE
db    0                          ; Indicates BIOS Structure Type
db    12h                        ; Length of information in bytes
dw    ?                          ; Reserved for handle
db    01h                        ; String 1 is the Vendor Name
db    02h                        ; String 2 is the BIOS version
dw    0E800h                     ; BIOS Starting Address
db    03h                        ; String 3 is the BIOS Build Date
dq    BIOS_Char                  ; BIOS Characteristics
db    1                          ; Size of BIOS ROM is 128K (64K * (1 + 1))
db    'System BIOS Vendor Name',0      ;
db    'Version 4.04',0           ;
db    '00/00/00',0               ;
db    0                          ; End of strings
```

3.2 Structure Definitions

3.2.1 BIOS Information (Type 0)

Offset	Name	Length	Value	Description
00h	Type	BYTE	0	BIOS Information Indicator
01h	Length	BYTE	Varies	12h + number of BIOS Characteristics Extension Bytes. If no Extension Bytes are used the Length will be 12h.
02h	Handle	WORD	Varies	
04h	Vendor	BYTE	Varies	String number of the BIOS Vendor's Name
05h	BIOS Version	BYTE	Varies	String number of the BIOS Version. This is a free form string which may contain Core and OEM version information.
06h	BIOS Starting Address Segment	WORD	Varies	Segment location of BIOS starting address, e.g.0E800h. **Note**: The size of the runtime BIOS image can be computed by subtracting the Starting Address Segment from 10000h and multiplying the result by 16.
08h	BIOS Release Date	BYTE	Varies	String number of the BIOS release date. The date string, if supplied, is in the format mm/dd/yy.
09h	BIOS ROM Size	BYTE	Varies (*n*)	Size (n) is 64K * (*n*+1)
0Ah	BIOS Characteristics	QWORD	Bit Field	Defines which functions the BIOS supports. PCI, PCMCIA, Flash, etc. See 3.2.1.1.
12h	BIOS Characteristics Extension Bytes	Zero or more BYTEs	Bit Field	Optional space reserved for future supported functions. The number of Extension Bytes that are present is indicated by the Length in offset 1 minus 12h.

3.2.1.1 BIOS Characteristics

QWORD Bit Position	Meaning if Set
Bit 0	Reserved
Bit 1	Reserved
Bit 2	Unknown
Bit 3	BIOS Characteristics Not Supported
Bit 4	ISA is supported
Bit 5	MCA is supported
Bit 6	EISA is supported
Bit 7	PCI is supported
Bit 8	PCMCIA is supported
Bit 9	Plug and Play is supported
Bit 10	APM is supported
Bit 11	BIOS is Upgradeable (Flash)
Bit 12	BIOS shadowing is allowed
Bit 13	VL-VESA is supported
Bit 14	ESCD support is available
Bit 15	Boot from CD is supported
Bit 16	Selectable Boot is supported
Bit 17	BIOS ROM is socketed
Bit 18	Boot From PCMCIA is supported
Bit 19	EDD (Enhanced Disk Drive) Specification is supported
Bit 20	Int 13h - Japanese Floppy for NEC 9800 1.2mb (3.5”, 1k Bytes/Sector, 360 RPM) is supported
Bit 21	Int 13h - Japanese Floppy for Toshiba 1.2mb (3.5”, 360 RPM) is supported
Bit 22	Int 13h - 5.25” / 360 KB Floppy Services are supported
Bit 23	Int 13h - 5.25” /1.2MB Floppy Services are supported
Bit 24	Int 13h - 3.5” / 720 KB Floppy Services are supported
Bit 25	Int 13h - 3.5” / 2.88 MB Floppy Services are supported
Bit 26	Int 5h, Print Screen Service is supported
Bit 27	Int 9h, 8042 Keyboard services are supported
Bit 28	Int 14h, Serial Services are supported
Bit 29	Int 17h, Printer Services are supported
Bit 30	Int 10h, CGA Video Services are supported
Bit 31	PC-98
Bits32:47	Reserved for BIOS Vendor
Bits 48:63	Reserved for System Vendor

3.2.1.2 BIOS Characteristics Extension Byte 1

BYTE Bit Position	Meaning if Set
Bits 7:0	Reserved, set to 0.

3.2.2 System Information (Type 1)

The information in this structure defines attributes of the overall system and is intended to be associated with the *Component ID* group of the system's MIF.

Offset	Name	Length	Value	Description
00h	Type	BYTE	1	Component ID Information Indicator
01h	Length	BYTE	08h	
02h	Handle	WORD	Varies	
04h	Manufacturer	BYTE	Varies	Number of Null terminated string
05h	Product Name	BYTE	Varies	Number of Null terminated string
06h	Version	BYTE	Varies	Number of Null terminated string
07h	Serial Number	BYTE	Varies	Number of Null terminated string

3.2.3 Base Board Information (Type 2)

The information in this structure defines attributes of the system's baseboard (also known as the motherboard or planar).

Offset	Name	Length	Value	Description
00h	Type	BYTE	2	Base Board Information Indicator
01h	Length	BYTE	08h	
02h	Handle	WORD	Varies	
04h	Manufacturer	BYTE	Varies	Number of Null terminated string
05h	Product	BYTE	Varies	Number of Null terminated string
06h	Version	BYTE	Varies	Number of Null terminated string
07h	Serial Number	BYTE	Varies	Number of Null terminated string

3.2.4 System Enclosure or Chassis (Type 3)

The information in this structure defines attributes of the system's mechanical enclosure(s). For example, if a system included a separate enclosure for its peripheral devices, two structures would be returned: one for the main, system enclosure and the second for the peripheral device enclosure.

Offset	Name	Length	Value	Description
00h	Type	BYTE	3	System Enclosure Indicator
01h	Length	BYTE	09h	
02h	Handle	WORD	Varies	
04h	Manufacturer	BYTE	Varies	Number of Null terminated string
05h	Type	BYTE	Varies	Bit 7 Chassis lock present if 1. Otherwise, either a lock is not present or it is unknown if the enclosure has a lock. Bits 6:0 Enumeration value, see below.
06h	Version	BYTE	Varies	Number of Null terminated string
07h	Serial Number	BYTE	Varies	Number of Null terminated string
08h	Asset Tag Number	BYTE	Varies	Number of Null terminated string

3.2.4.1 System Enclosure or Chassis Types

Byte Value	Meaning
01h	Other
02h	Unknown
03h	Desktop
04h	Low Profile Desktop
05h	Pizza Box
06h	Mini Tower
07h	Tower
08h	Portable
09h	LapTop
0Ah	Notebook
0Bh	Hand Held
0Ch	Docking Station
0Dh	All in One
0Eh	Sub Notebook
0Fh	Space-saving
10h	Lunch Box
11h	Main Server Chassis
12h	Expansion Chassis
13h	SubChassis
14h	Bus Expansion Chassis
15h	Peripheral Chassis
16h	RAID Chassis
17h	Rack Mount Chassis

3.2.5 Processor Information (Type 4)

The information in this structure defines the attributes of a single processor; a separate structure instance is provided for each system processor. For example, a system with an 80486DX processor would have a single structure instance while a system with an 80486SX processor would have a structure to describe the main CPU and a second structure to describe the 80487 co-processor.

Offset	Name	Length	Value	Description
00h	Type	BYTE	4	Processor Information Indicator
01h	Length	BYTE	1Ah	
02h	Handle	WORD	Varies	
04h	Socket Designation	BYTE	Varies	String number for Reference Designation. Example string 'J202',0
05h	Processor Type	BYTE	ENUM	See 3.2.5.1 on page 29
06h	Processor Family	BYTE	ENUM	See 3.2.5.2 on page 29
07h	Processor Manufacturer	BYTE	Varies	String number of Processor Manufacturer
08h	Processor ID	QWORD	Varies	Raw processor identification data. See 3.2.5.3 for details.
10h	Processor Version	BYTE	Varies	String number describing the Processor
11h	Voltage	BYTE	Varies	Bits 7:4 Reserved, must be zero Bits 3:0 Voltage Capability. A Set bit indicates the voltage is supported. Bit 0 - 5V Bit 1 - 3.3V Bit 2 - 2.9V Bit 3 - Reserved, must be zero. **Note**: Setting of multiple bits indicates the socket is configurable
12h	External Clock	WORD	Varies	External Clock Frequency. If the value is unknown, the field is set to 0.
14h	Max Speed	WORD	Varies	99d for a 99MHz processor. If the value is unknown, the field is set to 0.
16h	Current Speed	WORD	Varies	Same as Max Speed

Offset	Name	Length	Value	Description
18h	Status	BYTE	Varies	Bit 7 Reserved, must be 0 Bit 6 CPU Socket Populated 1 - CPU Socket Populated 0- CPU Socket Unpopulated Bits 5:3 Reserved, must be zero Bits 2:0 CPU Status 0h - Unknown 1h - CPU Enabled 2h - CPU Disabled by User (via BIOS Setup) 3h - CPU Disabled By System BIOS (POST Error) 4h - CPU is Idle (waiting to be Enabled) 5-6h - Reserved 7h - Other
19h	Processor Upgrade	BYTE	ENUM	See 3.2.5.4

3.2.5.1 Processor Information - Processor Type

Byte Value	Meaning
01h	Other
02h	Unknown
03h	Central Processor
04h	Math Processor
05h	DSP Processor
06h	Video Processor

3.2.5.2 Processor Information - Processor Family

Byte Value	Meaning
01h	Other
02h	Unknown
03h	8086
04h	80286
05h	80386
06h	80486
07h	8087
08h	80287
09h	80387
0Ah	80487
0Bh	Pentium Family
0Ch-11h	Reserved for specific Pentium versions
12h	M1 Family
13h-18h	Reserved for specific M1 versions
19h	K5 Family
1Ah-1Fh	Reserved for specific K5 versions
20h	Power PC Family
A0h	V30 Family

3.2.5.3 Processor ID Field Format

The Processor ID field contains processor-specific information which describes the processor's features.

3.2.5.3.1 X86-Class CPUs

For x86 class CPUs, the field's format depends on the processor's support of the CPUID instruction. If the instruction is supported, the *Processor ID* field contains two DWORD-formatted values. The first (offsets 08h-0Bh) is the EAX value returned by a CPUID instruction with input EAX set to 1; the second (offsets 0Ch-0Fh) is the EDX value returned by that instruction.

Otherwise, only the first two bytes of the *Processor ID* field are significant (all others are set to 0) and contain (in WORD-format) the contents of the DX register at CPU reset.

3.2.5.4 Processor Information - Processor Upgrade

Byte Value	Meaning
01h	Other
02h	Unknown
03h	Daughter Board
04h	ZIF Socket
05h	Replaceable Piggy Back
06h	None
07h	LIF Socket

3.2.6 Memory Controller Information (Type 5)

The information in this structure defines the attributes of the system's memory controller(s) and the supported attributes of any memory-modules present in the sockets controlled by this controller.

Offset	Name	Length	Value	Description
00h	Type	BYTE	5	Memory Controller Indicator
01h	Length	BYTE	Varies	15 + (2 X Number of Associated Memory Slots), offset 0Eh
02h	Handle	WORD	Varies	
04h	Error Detecting Method	BYTE	ENUM	See 3.2.6.1
05h	Error Correct-ing Capability	BYTE	Bit Field	See 3.2.6.2
06h	Supported Interleave	BYTE	ENUM	See 3.2.6.3
07h	Current Interleave	BYTE	ENUM	See 3.2.6.3
08h	Maximum Memory Module Size	BYTE	Varies (*n*)	The size of the largest memory module supported (per slot), specified as *n*, where *2**n* is the maximum size in MB. The maximum amount of memory supported by this controller is that value times the number of slots, as specified in offset 0Eh of this structure.
09h	Supported Speeds	WORD	Bit Field	See 3.2.6.4 for bit-wise descriptions.
0Bh	Supported Memory Types	WORD	Bit Field	See 3.2.7.1 on page 33 for bit-wise descriptions.
0Dh	Memory Module Voltage	BYTE	Varies	This field describes the required voltages for each of the memory module sockets controlled by this controller: Bits 7:3 Reserved, must be zero Bit 2 2.9V Bit 1 3.3V Bit 0 5V **Note**: Setting of multiple bits indicates the sockets are configurable
0Eh	Number of Associated Memory Slots	BYTE	Varies	Defines how many of the Memory Module Information blocks are controlled by this controller
0Fh+	Memory Module Configuration Handle	WORD	Varies	A memory information structure index controlled by this controller. Value in offset 0Eh defines the count.

3.2.6.1 Memory Controller Error Detecting Method

Byte Value	Meaning
01h	Other
02h	Unknown
03h	None
04h	8-bit Parity
05h	32-bit ECC
06h	64-bit ECC
07h	128-bit ECC

3.2.6.2 Memory Controller Error Correcting Capability

Byte Bit Position	Meaning
Bit 0	Other
Bit 1	Unknown
Bit 2	None
Bit 3	Single Bit Error Correcting
Bit 4	Double Bit Error Correcting
Bit 5	Error Scrubbing

3.2.6.3 Memory Controller Information - Interleave Support

Byte Value	Meaning
01h	Other
02h	Unknown
03h	One Way Interleave
04h	Two Way Interleave
05h	Four Way Interleave
06h	Eight Way Interleave
07h	Sixteen Way Interleave

3.2.6.4 Memory Controller Information - Memory Speeds

This bit-field describes the speed of the memory modules supported by the system.

Word Bit Position	Meaning
Bit 0	Other
Bit 1	Unknown
Bit 2	70ns
Bit 3	60ns
Bit 4	50ns
Bits 5:15	Reserved, must be zero

3.2.7 Memory Module Information (Type 6)

One *Memory Module Information* structure is included for each memory-module socket in the system. The structure describes the speed, type, size, and error status of each system memory module. The supported attributes of each module are described by the "owning" *Memory Controller Information* structure.

Offset	Name	Length	Value	Description
00h	Type	BYTE	6	Memory Module Configuration Indicator
01h	Length	BYTE	0Ch	
02h	Handle	WORD	Varies	
04h	Socket Designation	BYTE	Varies	String Number for Reference Designation. Example 'J202',0
05h	Bank Connections	BYTE	Varies	Each nibble indicates a bank (RAS#) connection, 0xF means no connection. Example: If banks 1 & 3 (RAS# 1 & 3) were connected to a SIMM socket the byte for that socket would be 13h. If only bank 2 (RAS 2) were connected the byte for that socket would be 2Fh.
06h	Current Speed	BYTE	Varies	The speed of the memory module, in ns (e.g. 70d for a 70ns module). If the speed is unknown, the field is set to 0.
07h	Current Memory Type	WORD	Bit Field	See 3.2.7.1
09h	Installed Size	BYTE	Varies	See 3.2.7.2
0Ah	Enabled Size	BYTE	Varies	See 3.2.7.2
0Bh	Error Status	BYTE	Varies	Bits 7:2 Reserved, set to 0's Bit 1 Correctable errors received for the module, if set Bit 0 Uncorrectable errors received for the module, if set. All or a portion of the module has been disabled.

3.2.7.1 Memory Module Information - Memory Types

This bit-field describes the physical characteristics of the memory modules which are supported by (and currently installed in) the system.

Word Bit Position	Meaning
Bit 0	Other
Bit 1	Unknown
Bit 2	Standard
Bit 3	Fast Page Mode
Bit 4	EDO
Bit 5	Parity
Bit 6	ECC
Bit 7	SIMM
Bit 8	DIMM
Bits 9:15	Reserved, must be zero

3.2.7.2 Memory Module Information - Memory Size

The Size fields of the Memory Module Configuration Information structure define the amount of memory currently installed (and enabled) in a memory-module connector.

The *Installed Size* fields identify the size of the memory module which is installed in the socket, as determined by reading and correlating the module's presence-detect information. If the system does not support presence-detect mechanisms, the *Installed Size* field is set to 7Dh to indicate that the installed size is not determinable. The *Enabled Size* field identifies the amount of memory currently enabled for the system's use from the module. If a module is known to be installed in a connector, but all memory in the module has been disabled due to error, the *Enabled Size* field is set to 7Eh.

Byte Bit Range	Meaning
Bits 0:6	Size (n), where 2**n is the size in MB with three special-case values: 7Dh Not determinable (Installed Size only) 7Eh Module is installed, but no memory has been enabled 7Fh Not installed
Bit 7	Defines whether the memory module has a single- (0) or double-bank (1) connection.

3.2.7.3 Memory Subsystem Example

A system utilizes a memory controller which supports up to 4-32MB 5V 70ns parity SIMMs. The memory module sockets are used in pairs A1/A2 and B1/B2 to provide a 64-bit data path to the CPU. No mechanism is provided by the system to read the SIMM IDs. RAS-0 and -1 are connected to the front- and back-size banks of the SIMMs in the A1/A2 sockets and RAS-2 and -3 are similarly connected to the B1/B2 sockets. The current installation is an 8MB SIMM in sockets A1 and A2, 16MB total.

```
db     5             ; Memory Controller Information
db     23            ; Length = 15 + 2*4
dw     14            ; Memory Controller Handle
db     4             ; 8-bit parity error detection
db     00000100b     ; No error correction provided
db     03h           ; 1-way interleave supported
db     03h           ; 1-way interleave currently used
db     5             ; Maximum memory-module size supported is 32MB (2**5)
dw     00000100b     ; Only 70ns SIMMs supported
dw     00A4h         ; Standard, parity SIMMs supported
db     00000001b     ; 5V provided to each socket
db     4             ; 4 memory-module sockets supported
dw     15            ; 1st Memory Module Handle
dw     16
dw     17
dw     18            ; 4th ...
```

```
db     6              ; Memory Module Information
db     0Ch
dw     15             ; Handle
db     1              ; Reference Designation string #1
db     01h            ; Socket connected to RAS-0 and RAS-1
db     00000010b      ; Current speed is Unknown, since can't read SIMM IDs
db     00000100b      ; Upgrade speed is 70ns, since that's all that's
                      ; supported
dw     00A4h          ; Current SIMM must be standard parity
db     7Dh            ; Installed size indeterminable (no SIMM IDs)
db     83h            ; Enabled size is double-bank 8MB (2**3)
db     0              ; No errors
db     "A1",0         ; String#1: Reference Designator
db     0              ; End-of-strings
db     6              ; Memory Module Information
db     0Ch
dw     16             ; Handle
db     1              ; Reference Designation string #1
db     01h            ; Socket connected to RAS-0 and RAS-1
db     0              ; Current speed is Unknown, since can't read SIMM IDs
dw     00A4h          ; Current SIMM must be standard parity
db     7Dh            ; Installed size indeterminable (no SIMM IDs)
db     83h            ; Enabled size is double-bank 8MB (2**3)
db     0              ; No errors
db     "A2",0         ; String#1: Reference Designator
db     0              ; End-of-strings
db     6              ; Memory Module Information
db     0Ch
dw     17             ; Handle
db     1              ; Reference Designation string #1
db     23h            ; Socket connected to RAS-2 and RAS-3
db     0              ; Current speed is Unknown, since can't read SIMM IDs
dw     0001h          ; Nothing appears to be installed (Other)
db     7Dh            ; Installed size indeterminable (no SIMM IDs)
db     7Fh            ; Enabled size is 0 (nothing installed)
db     0              ; No errors
db     "B1",0         ; String#1: Reference Designator
db     0              ; End-of-strings

db     6              ; Memory Module Information
db     0Ch
dw     18             ; Handle
db     1              ; Reference Designation string #1
db     23h            ; Socket connected to RAS-2 and RAS-3
db     0              ; Current speed is Unknown, since can't read SIMM IDs
dw     0001h          ; Nothing appears to be installed (Other)
db     7Dh            ; Installed size indeterminable (no SIMM IDs)
db     7Fh            ; Enabled size is 0 (nothing installed)
db     0              ; No errors
db     "B2",0         ; String#1: Reference Designator
db     0              ; End-of-strings
```

3.2.8 Cache Information (Type 7)

The information in this structure defines the attributes of CPU cache device in the system. One structure is specified for each such device, whether the device is internal to or external to the CPU module. Cache modules can be associated with a processor structure, see 3.2.15 Group Associations (Type 14) on page 44 for more information.

Offset	Name	Length	Value	Description
00h	Type	BYTE	7	Cache Information Indicator
01h	Length	BYTE	0Fh	
02h	Handle	WORD	Varies	
04h	Socket Designation	BYTE	Varies	String Number for Reference Designation Example: “CACHE1”, 0
05h	Cache Configuration	WORD	Varies	Bits 15:10 Reserved, must be zero Bits 9:8 Operational Mode 00b Write Through 01b Write Back 10b Varies with Memory Address 11b Unknown Bit 7 Enabled/Disabled (at boot time) 1b Enabled 0b Disabled Bits 6:5 Location, relative to the CPU module: 00b Internal 01b External 10b Reserved 11b Unknown Bit 4 Reserved, must be zero Bit 3 Cache Socketed 1b Socketed 0b Not Socketed Bits 2:0 Cache Level - 1 through 8
07h	Maximum Cache Size	WORD	Varies	Maximum size that can be installed Bit 15 Granularity 0 - 1K granularity 1 - 64K granularity Bits 14:0 Max size in given granularity
09h	Installed Size	WORD	Varies	Same as Max Cache Size field
0Bh	Supported SRAM Type	WORD	Bit Field	See 3.2.8.1
0Dh	Current SRAM Type	WORD	Bit Field	See 3.2.8.1

3.2.8.1 Cache Information - SRAM Type

Word Bit Position	Meaning
Bit 0	Other
Bit 1	Unknown
Bit 2	Non-Burst
Bit 3	Burst
Bit 4	Pipeline Burst
Bit 5	Synchronous
Bit 6	Asynchronous
Bits 7:15	Reserved, must be zero

3.2.9 Port Connector Information (Type 8)

The information in this structure defines the attributes of a system port connector, e.g. parallel, serial, keyboard, mouse ports. The port's type and connector information are provided. One structure is present for each port provided by the system.

Offset	Name	Length	Value	Description
00h	Type	BYTE	8	Connector Information Indicator
01h	Length	BYTE	9h	
02h	Handle	WORD	Varies	
04h	Internal Reference Designator	BYTE	Varies	String number forInternal Reference Designator, i.e. internal to the system enclosure, e.g. 'J101', 0
05h	Internal Connector Type	BYTE	ENUM	Internal Connector type. See 3.2.9.2
06h	External Reference Designator	BYTE	Varies	String number for the External Reference Designation external to the system enclosure, e.g. 'COM A', 0
07h	External Connector Type	BYTE	ENUM	External Connector type. See 3.2.9.2
08h	Port Type	BYTE	ENUM	Describes the function of the port. See 3.2.9.3

3.2.9.1 Port Information Example

The following structure shows an example where a DB-9 Pin Male connector on the System Backpanel (COM A) is connected to the System Board via a 9 Pin Dual Inline connector (J101).

```
db     8              ; Indicates Connector Type
db     9h             ; Length
dw     ?              ; Reserved for handle
db     01h            ; String 1 - Internal Reference Designation
db     18h            ; 9 Pin Dual Inline
db     02h            ; String 2 - External Reference Designation
db     08h            ; DB-9 Pin Male
db     09h            ; 16550A Compatible
db     'J101',0       ; Internal reference
db     'COM A',0      ; External reference
db     0
```

If an External Connector is not used (as in the case of a CD-ROM Sound connector) then the External Reference Designator and the External Connector type should be set to zero. If an Internal Connector is not used (as in the case of a soldered on Parallel Port connector which extends outside of the chassis) then the Internal Reference Designation and Connector Type should be set to zero.

3.2.9.2 Port Information - Connector Types

Byte Value	Meaning
00h	None
01h	Centronics
02h	Mini Centronics
03h	Proprietary
04h	DB25 pin male
05h	DB25 pin female
06h	DB-15 pin male
07h	DB-15 pin female
08h	DB-9 pin male
09h	DB9 pin female
0Ah	RJ-11
0Bh	RJ-45
0Ch	50 Pin MiniSCSI
0Dh	Mini-DIN
0Eh	Micro-DIN
0Fh	PS/2
10h	Infrared
11h	HP-HIL
12h	Access Bus
13h	SSA SCSI
14h	Circular DIN-8 male
15h	Circular DIN-8 female
16h	On Board IDE
17h	On Board Floppy
18h	9 Pin Dual Inline (pin 10 cut)
19h	25 Pin Dual Inline (pin 26 cut)
1Ah	50 Pin Dual Inline
1Bh	68 Pin Dual Inline
1Ch	On Board Sound Input from CD-ROM
1Dh	Mini-Centronics Type-14
1Eh	Mini-Centronics Type-26
A0h	PC-98
A1h	PC-98Hireso
A2h	PC-H98
A3h	PC-98Note
A4h	PC-98Full
FFh	Other - Use Reference Designator Strings to supply information.

3.2.9.3 Port Types

Byte Value	Meaning
00h	None
01h	Parallel Port XT/AT Compatible
02h	Parallel Port PS/2
03h	Parallel Port ECP
04h	Parallel Port EPP
05h	Parallel Port ECP/EPP
06h	Serial Port XT/AT Compatible
07h	Serial Port 16450 Compatible
08h	Serial Port 16550 Compatible
09h	Serial Port 16550A Compatible
0Ah	SCSI Port
0Bh	MIDI Port
0Ch	Joy Stick Port
0Dh	Keyboard Port
0Eh	Mouse Port
0Fh	SSA SCSI
10h	USB
11h	FireWire (IEEE P1394)
12h	PCMCIA Type II
13h	PCMCIA Type II
14h	PCMCIA Type III
15h	Cardbus
16h	Access Bus Port
17h	SCSI II
18h	SCSI Wide
19h	PC-98
1Ah	PC-98-Hireso
1Bh	PC-H98
A0h	8251 Compatible
A1h	8251 FIFO Compatible
0FFh	Other

3.2.10 System Slots (Type 9)

The information in this structure defines the attributes of a system slot. One structure is provided for each slot in the system.

Offset	Name	Length	Value	Description
00h	Type	BYTE	9	System Slot Structure Indicator
01h	Length	BYTE	0Ch	
02h	Handle	WORD	Varies	
04h	Slot Designation	BYTE	Varies	String number for reference designation e.g. 'PCI-1',0
05h	Slot Type	BYTE	ENUM	See 3.2.10.1
06h	Slot Data Bus Width	BYTE	ENUM	See 3.2.10.2
07h	Current Usage	BYTE	BYTE	See 3.2.10.3
08h	Slot Length	BYTE	ENUM	See 3.2.10.4
09h	Slot ID	WORD	Varies	See 3.2.10.5
0Bh	Slot Characteristics	BYTE	Bit Field	See 3.2.10.6

3.2.10.1 System Slots - Slot Type

Byte Value	Meaning
01h	Other
02h	Unknown
03h	ISA
04h	MCA
05h	EISA
06h	PCI
07h	PCMCIA
08h	VL-VESA
09h	Proprietary
0Ah	Processor Card Slot
0Bh	Proprietary Memory Card Slot
0Ch	I/O Riser Card Slot
0Dh	NuBus
0Eh	PCI - 66MHz Capable
A0h	PC-98/C20
A1h	PC/98/C24
A2h	PC-98/E
A3h	PC-98/Local Bus
A4h	PC-98/Card

3.2.10.2 System Slots - Slot Data Bus Width

Byte Value	Meaning
01h	Other
02h	Unknown
03h	8 bit
04h	16 bit
05h	32 bit
06h	64 bit
07h	128 bit

3.2.10.3 System Slots - Current Usage

Byte Value	Meaning
01h	Other
02h	Unknown
03h	Available
04h	In use

3.2.10.4 System Slots - Slot Length

Byte Value	Meaning
01h	Other
02h	Unknown
03h	Half Length
04h	Full Length

3.2.10.5 System Slots — Slot ID

The *Slot ID* field of the System Slot structure provides a mechanism to correlate the physical attributes of the slot to its logical access method (which varies based on the *Slot Type* field). The Slot ID field has meaning only for the slot types described below:

Slot Type	Slot ID Field Meaning
MCA	Identifies the logical Micro Channel slot number, in the range 1 to 15, in offset 09h. Offset 0Ah is set to 0.
EISA	Identifies the logical EISA slot number, in the range 1 to 15, in offset 09h. Offset 0Ah is set to 0.
PCI	Identifies the value present in the Slot Number field of the PCI Interrupt Routing table entry that is associated with this slot, in offset 09h — offset 0Ah is set to 0. The table is returned by the "Get PCI Interrupt Routing Options" BIOS function call. *Note*: This definition also applies to the 66MHz-capable PCI slots.
PCMCIA	Identifies the Adapter Number (offset 09h) and Socket Number (offset 0Ah) to be passed to PCMCIA Socket Services to identify this slot.

3.2.10.6 Slot Characteristics

BYTE Bit Position	Meaning if Set
Bit 0	Characteristics Unknown
Bit 1	Provides 5.0 Volts
Bit 2	Provides 3.3 Volts
Bit 3	Slot's opening is shared with another slot, e.g. PCI/EISA shared slot.
Bits 4:7	Reserved, must be 0

3.2.11 On Board Devices Information (Type 10)

The information in this structure defines the attributes of devices which are onboard (soldered onto) a system element, usually the baseboard.

Offset	Name	Length	Value	Description
00h	Type	BYTE	10	On Board Devices Information Indicator
01h	Length	BYTE	Varies	4 + (Number of Devices x 2)
02h	Handle	WORD	Varies	
04h	Device 1 Type	BYTE	Varies	Bit 7 Device 1 Status 1 - Device Enabled 0 - Device Disabled Bits 6:0 Type of Device (See 3.2.11.1)
05h	Description String	BYTE	Varies	String number of description
06h	Device 2 Type	BYTE	Varies	Bit 7 Device 2 Status 1 - Device Enabled 0 - Device Disabled Bits 6:0 Type of Device (See 3.2.11.1)
07h	Description String	BYTE	Varies	String number of description

Note: This structure may contain the information on all onboard devices or there may be multiple instances of it for multiple devices.

3.2.11.1 Onboard Device Types

Byte Value	Meaning
01h	Other
02h	Unknown
03h	Video
04h	SCSI
05h	Ethernet
06h	Token Ring
07h	Sound

3.2.12 OEM Strings (Type 11)

Offset	Name	Length	Value	Description
00h	Type	BYTE	11	OEM Strings Indicator
01h	Length	BYTE	5h	
02h	Handle	WORD	Varies	
04h	Count	BYTE	Varies	Number of strings

This structure contains free form strings defined by the OEM. Examples of this are: Part Numbers for Reference Documents for the system, contact information for the manufactuer, etc.

3.2.13 System Configuration Options (Type 12)

Offset	Name	Length	Value	Description
00h	Type	BYTE	12	Configuration Information Indicator
01h	Length	BYTE	5h	
02h	Handle	WORD	Varies	
04h	Count	BYTE	Varies	Number of strings

This structure contains information required to configure the base board's Jumpers and Switches.
Examples of this are: "JP2: 1-2 Cache Size is 256K, 2-3 Cache Size is 512K"
"SW1-1: Close to Disable On Board Video"

3.2.14 BIOS Language Information (Type 13)

The information in this structure defines the installable language attributes of the BIOS. If the structure is not included within the BIOS DMI Information, the BIOS supports only English.

Offset	Name	Length	Value	Description
00h	Type	BYTE	13	Language Information Indicator
01h	Length	BYTE	16h	
02h	Handle	WORD	Varies	
04h	Installable Languages	BYTE	Varies	Number of languages available. Each available language will have a description string. i.e. "English". This field contains the number of strings that follow.
05h	Reserved	16 BYTEs	0	Reserved for future use
015h	Current Language	BYTE	Varies	String number (one-based) of the currently installed language.

The strings describing the languages follow the Current Language byte. The format of the strings is:
"ISO 639 Language Name | ISO 3166 Territory Name | Encoding Method". See the example below.

Note: Refer to the Desktop Management Interface Specification, V1.0, Appendix A (ISO 639) and Appendix B (ISO 3166) for additional information.

3.2.14.1 Example: BIOS Language Information

```
db     13                          ; language information
db     16h                         ; length
dw     ??                          ; handle
db     3                           ; three languages available
db     16 dup (0)                  ; reserved
db     2                           ; current language is French Canadian
db     'en|US|iso8859-1'           ; language 1 is US English
db     'fr|CA|iso8859-1'           ; language 2 is French Canadian
db     'ja|JP|unicode'             ; language 3 is Japanese
```

3.2.15 Group Associations (Type 14)

Offset	Name	Length	Value	Description
00h	Type	BYTE	14	Group Associations Indicator
01h	Length	BYTE	Varies	5 + (3 bytes for each item in the group)
02h	Handle	WORD	Varies	
04h	Group Name	BYTE	Varies	String number of string describing the group
05h	Item Type	BYTE	Varies	Item (Structure) Type of this member
06h	Item Handle	WORD	Varies	Handle corresponding to this structure

The Group Associations structure is provided for OEMs who want to specify the arrangement or hierarchy of certain components (including other Group Associations) within the system. For example, you can use the Group Associations structure to indicate that two CPU's share a common external cache system. These structures might look as follows:

First Group Association Structure:

```
db     14     ; Group Association structure
db     11     ; Length
dw     28h    ; Handle
db     01h    ; String Number (First String)
db     04     ; CPU Structure
dw     08h    ; CPU Structure's Handle
db     07     ; Cache Structure
dw     09h    ; Cache Structure's Handle
db     'Primary CPU Module', 0
db     0
```

Second Group Association Structure:

```
db     14     ; Group Association structure
db     11     ; Length
dw     29h    ; Handle
db     01h    ; String Number (First String)
db     04     ; CPU Structure
dw     0Ah    ; CPU Structure's Handle
db     07     ; Cache Structure
dw     09h    ; Cache Structure's Handle
db     'Secondary CPU Module', 0
db     0
```

In the examples above, CPU structures 08h and 0Ah are associated with the same cache, 09h. This relationship could also be specified as a single group:

```
db     14     ; Group Association structure
db     14     ; Length (5 + 3 * 3)
dw     28h    ; Structure handle for Group Association
db     1      ; String Number (First string)
db     4      ; 1st CPU
dw     08h    ; CPU structure handle
db     4      ; 2nd CPU
dw     0Ah    ; CPU structure handle
db     7      ; Shared cache
dw     09h    ; Cache structure handle
db     'Dual-Processor CPU Complex', 0
db     0
```

3.2.16 System Event Log (Type 15)

The presence of this structure within the DMI data returned for a system indicates that the system supports an event log. An event log is a fixed-length area within a non-volatile storage element, starting with a fixed-length (and vendor-specific) header record, followed by one or more variable-length log records. See 3.2.16.3 Event Log Organization on page 48 for more information. Refer also to *2.6 Function 54h – DMI Control* on page 15 for interfaces which can be used to control the event-log.

An application can implement event-log change notification by periodically reading the System Event Log structure (via its assigned handle) looking for a change in the *Log Change Token*. This token uniquely identifies the last time the event log was updated. When it sees the token changed, the application can retrieve the entire event log and determine the changes since the last time it read the event log.

Offset	Name	Length	Value	Description
00h	Type	BYTE	15	Event Log Type Indicator
01h	Length	BYTE	14h	Length of the structure, including the *Type* and *Length* fields..
02h	Handle	WORD	Var	The handle, or instance number, associated with the record.
04h	Log Area Length	WORD	Var	The length, in bytes, of the overall event log area, from the first byte of header to the last byte of data.
06h	Log Header Start Offset	WORD	Var	Defines the starting offset (or index) within the nonvolatile storage of the event-log's header, from the *Access Method Address*. For single-byte indexed I/O accesses, the most-significant byte of the start offset is set to 00h. If the log area has no header, this field is set to 0.
08h	Log Data Start Offset	WORD	Var	Defines the starting offset (or index) within the nonvolatile storage of the event-log's first data byte, from the *Access Method Address*. For single-byte indexed I/O accesses, the most-significant byte of the start offset is set to 00h. **Note**: The data directly follows any header information. Therefore, the header length can be determined by subtracting the *Header Start Offset* from the *Data Start Offset*.

Offset	Name	Length	Value	Description
0Ah	Access Method	BYTE	Var	Defines the Location and Method used by higher-level software to access the log area, one of: 00h Indexed I/O: 1 8-bit index port, 1 8-bit data port. The *Access Method Address* field contains the 16-bit I/O addresses for the index and data ports. See 3.2.16.1 for usage details. 01h Indexed I/O: 2 8-bit index ports, 1 8-bit data port. The *Access Method Address* field contains the 16-bit I/O address for the index and data ports. See 3.2.16.1 for usage details. 02h Indexed I/O: 1 16-bit index port, 1 8-bit data port. The *Access Method Address* field contains the 16-bit I/O address for the index and data ports. See 3.2.16.1 for usage details. 03h Memory-mapped physical 32-bit address. The *Access Method Address* field contains the 4-byte (Intel DWORD format) starting physical address. 04h Available via *General-Purpose NonVolatile Data* functions, see 2.7 on page 17 for more information. The Access Method Address field contains the 2-byte (Intel WORD format) GPNV handle. 05h-FFh Available for future assignment
0Bh	Log Status	BYTE	Var	This bit-field describes the current status of the system event-log: Bits 7:2 Reserved, set to 0's Bit 1 Log area full, if 1 Bit 0 Log area valid, if 1
0Ch	Log Change Token	DWORD	Var	Unique token that is reassigned every time the event log changes. Can be used to determine if additional events have occurred since the last time the log was read.

Offset	Name	Length	Value	Description

10h	Access Method Address	DWORD	Var	The address associated with the access method; the data present depends on the *Access Method* field value. The area's format can be described by the following 1-byte-packed 'C' union:
				`union`
				`{`
				`struct`
				`{`
				`short IndexAddr;`
				`short DataAddr;`
				`} IO;`
				`long PhysicalAddr32;`
				`short GPNVHandle;`
				`} AccessMethodAddress;`

3.2.16.1 Indexed I/O Access Method

This section contains examples (in x86 assembly language) which detail the code required to access the "indexed I/O" event-log information.

3.2.16.1.1 1 8-bit Index, 1 8-bit Data (00h)

To access the event-log, the caller selects 1 of 256 unique data bytes by

1) Writing the byte data-selection value (index) to the *IndexAddr* I/O address
2) Reading or writing the byte data value to (or from) the *DataAddr* I/O address

```
mov    dx, IndexAddr       ;Value from event-log structure
mov    al, WhichLoc        ;Identify offset to be accessed
out    dx, al
mov    dx, DataAddr        ;Value from event-log structure
in     al, dx              ; Read current value
```

3.2.16.1.2 2 8-bit Index, 1 8-bit Data (01h)

To access the event-log, the caller selects 1 of 65536 unique data bytes by

1) Writing the least-significant byte data-selection value (index) to the *IndexAddr* I/O address
2) Writing the most-significant byte data-selection value (index) to the (*IndexAddr+1*) I/O address
3) Reading or writing the byte data value to (or from) the *DataAddr* I/O address

```
mov    dx, IndexAddr       ;Value from event-log structure
mov    ax, WhichLoc        ;Identify offset to be accessed
out    dx, al              ;Select LSB offset
inc    dx
xchg   ah, al
out    dx, al              ;Select MSB offset
mov    dx, DataAddr        ;Value from event-log structure
in     al, dx              ;Read current value
```

3.2.16.1.3 1 16-bit Index, 1 8-bit Data (02h)

To access the event-log, the caller selects 1 of 65536 unique data bytes by

1) Writing the word data-selection value (index) to the *IndexAddr* I/O address
2) Reading or writing the byte data value to (or from) the *DataAddr* I/O address

```
mov    dx, IndexAddr       ;Value from event-log structure
mov    ax, WhichLoc        ;Identify offset to be accessed
out    dx, ax
mov    dx, DataAddr        ;Value from event-log structure
in     al, dx              ;Read current value
```

3.2.16.2 Access Method Address — DWORD Layout

Access Type	BYTE 3	BYTE 2	BYTE 1	BYTE 0
00:02 — Indexed I/O	Data MSB	Data LSB	Index MSB	Index LSB
03- Absolute Address	Byte 3	Byte 2	Byte 1	Byte 0
04 - Use GPNV	0	0	Handle MSB	Handle LSB

3.2.16.3 Event Log Organization

The event log is organized as an optional (and implementation-specific) fixed-length header, followed by one or more variable-length event records, as illustrated below1. From one implementation to the next, the format of the log header and the size of the overall log area might change; all other fields of the event log area will be consistent across all systems. *1*

Log Header (Optional)								
Type	Length	Year	Month	Day	Hour	Minute	Second	Log Variable Data
Reqd	**Reqd**	**Reqd**	**Reqd**	**Reqd**	**Reqd**	**Reqd**	**Reqd**	Optional

3.2.16.4 Log Record

Each log record consists of a *required* fixed-length leader, followed by (optional) additional data which is defined by the event type. The fixed-length log record header is present as the first 8-bytes of each log record, regardless of event type, and consists of:

Offset	Name	Format	Description
00h	Event Type	BYTE	Specifies the "Type" of event noted in an event-log entry as defined in 3.2.16.4.1
01h	Length	BYTE	Specifies the byte length of the event record, including the record's Type and Length fields. The most-significant bit of the field specifies whether (0) or not (1) the record has been read. The implication of the record having been read is that the information in the log record has been processed by a higher software layer.
02h-07h	Date/Time Fields	BYTE	These fields contain the BCD representation of the date and time (as read from CMOS) of the most recent occurrence of the event. The information is present in year, month, day, hour, minute, second order.
08h+	Log Variable Data	Var	This field contains the (optional) event-specific additional status information.

3.2.16.4.1 Event Log Types

Value	Description
00h	Reserved.
01h	Single-bit ECC memory error
02h	Multi-bit ECC memory error
03h	Parity memory error
04h	Bus time-out
05h	I/O Channel Check
06h	Software NMI
07h	POST Memory Resize
08h	POST Error
09h	PCI Parity Error
0Ah	PCI System Error
0Bh	CPU Failure
0Ch	EISA FailSafe Timer time-out
0Dh	Correctable memory log disabled
0Eh	Logging disabled for a specific Event Type – too many errors of the same type received in a short amount of time.
0Fh	Reserved
10h	System Limit Exceeded (e.g. voltage or temperature threshold exceeded).
11h	Asynchronous hardware timer expired and issued a system reset.
12h	System configuration information
13h	Hard-disk information
14h	System reconfigured
15h	Uncorrectable CPU-complex error
16h	Log Area Reset/Cleared
17h	System boot. If implemented, this log entry is guaranteed to be the first one written on any system boot.
18h-7Fh	Unused, available for assignment by this specification.
80h-FFh	Available for system- and OEM-specific assignments.