[UEFI Practice] BIOS and IPMI

Category Column: BMC   UEFI Development Basics   Article Tags: uefi   ipmi

---

2048 AI Community   The article has been collected by the community   Join the community

UEFI Development …   Included in 2 columns at the same time ▾   136 articles   Subscribe to our column

---

⚠摘要   KCS is the abbreviation of KeyboardControllerStyle, which is the basic communication method between BIOS and BMC. This article introduces the KCS interface in detail, including the data structure of Request and Response, and the type of KCS data. In addition, the IO registers and communication process of the KCS interface are analyzed, especially the status and control code of the Status register. Finally, the application of IPMI commands in the KCS interface is mentioned, as well as the command descriptions under different NetFn.

The summary is generated in C Know , supported by DeepSeek-R1 full version, go to experience>

KCS

KCS stands for **Keyboard Controler Style** . You don't need to go into too much detail about this name. You just need to know that it is a basic way for the system (BIOS and OS) to communicate with BMC. This article will introduce the KCS interface under BIOS, including the interface usage and data. The content is referenced from "ipmi-second-gen-interface-spec-v2-rev1-1.pdf" and https://github.com/microsoft/mu_feature_ipmi.git code.

KCS Data

The interaction between BIOS and BMC is through Request and Response, so there are two kinds of corresponding data. The first is Request:

Figure 9-, KCS Interface/BMC Request Message Format

Byte 1	Byte 2	Byte 3:N
NetFn/LUN	Cmd	Data

The parameters are described as follows:

- **NetFn/Cmd** : IPMI commands are grouped in different ways and indexed by NetFn/Cmd command words, which will be further explained in the IPMI command introduction .
- **LUN** : **Logical Unit** Number , occupies the lowest two **bits** of the 8 bits. For data sent to the BMC through the KCS, the LUN value is 00b .
- **Data** : The actual request data of the IPMI command.

Then the Response:

Figure 9-, KCS Interface/BMC Response Message Format

Byte 1	Byte 2	Byte 3	Byte 4:N
NetFn/LUN	Cmd	Completion Code	Data

The other parameters are similar to Request, except for an additional Complete Code, as described below:

- **Completion Code** : The return status of the IPMI command. The specific values are as follows:

```c
// AI generated projects    登录复制    run
// Generic Completion Codes definitions
//
#define IPMI_COMP_CODE_NORMAL                          0x00
#define IPMI_COMP_CODE_NODE_BUSY                       0xC0
#define IPMI_COMP_CODE_INVALID_COMMAND                 0xC1
#define IPMI_COMP_CODE_INVALID_FOR_GIVEN_LUN           0xC2
#define IPMI_COMP_CODE_TIMEOUT                         0xC3
#define IPMI_COMP_CODE_OUT_OF_SPACE                    0xC4
#define IPMI_COMP_CODE_RESERVATION_CANCELED_OR_INVALID 0xC5
#define IPMI_COMP_CODE_REQUEST_DATA_TRUNCATED          0xC6
#define IPMI_COMP_CODE_INVALID_REQUEST_DATA_LENGTH     0xC7
#define IPMI_COMP_CODE_REQUEST_EXCEED_LIMIT            0xC8
#define IPMI_COMP_CODE_OUT_OF_RANGE                    0xC9
#define IPMI_COMP_CODE_CANNOT_RETURN                   0xCA
#define IPMI_COMP_CODE_NOT_PRESENT                     0xCB
#define IPMI_COMP_CODE_INVALID_DATA_FIELD              0xCC
#define IPMI_COMP_CODE_COMMAND_ILLEGAL                 0xCD
#define IPMI_COMP_CODE_CMD_RESP_NOT_PROVIDED           0xCE
#define IPMI_COMP_CODE_FAIL_DUP_REQUEST                0xCF
#define IPMI_COMP_CODE_SDR_REP_IN_UPDATE_MODE          0xD0
#define IPMI_COMP_CODE_DEV_IN_FW_UPDATE_MODE           0xD1
#define IPMI_COMP_CODE_BMC_INIT_IN_PROGRESS            0xD2
#define IPMI_COMP_CODE_DEST_UNAVAILABLE                0xD3
#define IPMI_COMP_CODE_INSUFFICIENT_PRIVILEGE          0xD4
#define IPMI_COMP_CODE_UNSUPPORTED_IN_PRESENT_STATE    0xD5
#define IPMI_COMP_CODE_SUBFUNCTION_DISABLED            0xD6
#define IPMI_COMP_CODE_UNSPECIFIED                     0xFF
```

收起 ∧

- **Data** : The actual response data of the IPMI command.

It should be noted that the data part of both Request and Response can be empty.

In addition, there is a slightly special KSC data, which is used to report event information:

Figure 9-, KCS Interface Event Request Message Format

NetFn (04h = Sensor/Event Request)		LUN (00b)	Command (02h = Platform Event)		Software ID (Gen ID) 7-bits		1
EvMRev	Sensor Type	Sensor #	Event Dir	Event Type	Event Data 1	Event Data 2	Event Data 3

Shading designates fields that are not stored in the event record.

Figure 9-, KCS Interface Event Response Message Format

NetFn (05h = Sensor/Event Response)	00	Command (02h = Platform Event)	Completion Code

In fact, it is just a kind of IPMI command data.

KSC Communications

The KSC interface defines a set of IO registers through which the interaction with the BMC can be completed. The base address of this set of registers is basically fixed:

```
gIpmiFeaturePkgTokenSpaceGuid.PcdIpmiIoBaseAddress|0xCA2|UINT16|0xF000000A
```

In order to use this address, you first need to open its mapping, as shown below:

```c
/**
  Initializing hardware for the IPMI transport.

  @retval   EFI_SUCCESS     Hardware was successfully initialized.
  @retval   Other           An error was returned from PlatformIpmiIoRangeSet.
**/
EFI_STATUS
InitializeIpmiTransportHardware (
  VOID
  )

{
  EFI_STATUS  Status;

  //
  // Enable OEM specific southbridge SIO KCS I/O address range 0xCA0 to 0xCAF at here
  // if the the I/O address range has not been enabled.
  //
  Status = PlatformIpmiIoRangeSet (PcdGet16 (PcdIpmiIoBaseAddress));
  DEBUG ((DEBUG_INFO, "IPMI: PlatformIpmiIoRangeSet - %r!\n", Status));
  return Status;
}
```

As for `PlatformIpmiIoRangeSet()` the implementation of the function, you can temporarily ignore it because it depends on different hardware platforms. For the x86 platform, it is usually some operations of the LPC device.

After completing `PcdIpmiIoBaseAddress` the base address, you can perform IO operations. The corresponding registers are as follows:

Figure 9-, KCS Interface Registers

	7	6	5	4	3	2	1	0	I/O address
Status (ro)	S1	S0	OEM 2	OEM 1	C/D#	SMS_ATN	IBF	OBF	base+1
Command (wo)									base+1
Data_Out (ro)									base+0
Data_In (wo)									base+0

Reserved bits must be written as '0' and ignored during reads. Software should not assume that reserved bits return a constant value.

You can see that there are only 4 registers (if the shared part is considered, there are actually only 2 registers):

- **Status** : Read-only register, contains the flag during the operation.
- **Command** : Write-only register, used to write various operations, which are called "Write Control Codes".
- **Data_Out** : Read-only register, used to read data.
- **Data_In** : Write-only register, used to write data or "Read Control Codes".

The "Control Codes" here are as follows:

Table 9-, KCS Interface Control Codes

Code	Name	Description	Target register	Output Data Register
60h	GET_STATUS / ABORT	Request Interface Status / Abort Current operation	Command	Status Code
61h	WRITE_START	Write the First byte of an Write Transfer	Command	N/A.
62h	WRITE_END	Write the Last byte of an Write Transfer	Command	N/A
63h-67h	reserved	reserved		
68h	READ	Request the next data byte	Data_In	Next byte
69h-6Fh	reserved	reserved		

The first four are "Write Control Codes", which can only be written to the Status register; the last one is "Read Control Codes", which is written to the Data register. The KSC data mentioned above is also written to the Data register.

The bits of the Status register are described as follows:

Table 9-, KCS Interface Status Register Bits

Bit	Name	Description	R/W[1]
7	S1	State bit 1. Bits 7 & 6 are used to indicate the current state of the KCS Interface. Host Software should examine these bits to verify that it's in sync with the BMC. See below for more detail.	R/O
6	S0	State bit 0. See bit 7.	R/O
5	OEM2	OEM - reserved for BMC implementer / system integrator definition.	R/O
4	OEM1	OEM - reserved for BMC implementer / system integrator definition.	R/O
3	C/D#	Specifies whether the last write was to the Command register or the Data_In register (1=command, 0=data). Set by hardware to indicate whether last write from the system software side was to Command or Data_In register.	R/O
2	SMS_ATN	Set to 1 when the BMC has one or more messages in the Receive Message Queue, or when a watchdog timer pre-timeout, or event message buffer full condition exists[2]. OEMs may also elect to set this flag is one of the OEM 1, 2, or 3 flags from the *Get Message Flags* command becomes set. This bit is related to indicating when the BMC is the source of a system interrupt. Refer to sections *9.12, KCS Communication and Non-communication Interrupts, 9.13, Physical Interrupt Line Sharing,* and *9.14, Additional Specifications for the KCS interface* for additional information on the use and requirements for the SMS_ATN bit.	R/O
1	IBF	Automatically set to 1 when either the associated Command or Data_In register has been written by system-side software.	R/O
0	OBF	Set to 1 when the associated Data_Out register has been written by the BMC.	R/O

The values of S0 and S1 are as follows:

Table 9-, KCS Interface State Bits

S1 (bit 7)	S0 (bit 6)	Definition
0	0	IDLE_STATE. Interface is idle. System software should not be expecting nor sending any data.
0	1	READ_STATE. BMC is transferring a packet to system software. System software should be in the "Read Message" state.
1	0	WRITE_STATE. BMC is receiving a packet from system software. System software should be writing a command to the BMC.
1	1	ERROR_STATE. BMC has detected a protocol violation at the interface level, or the transfer has been aborted. System software can either use the "Get_Status' control code to request the nature of the error, or it can just retry the command.

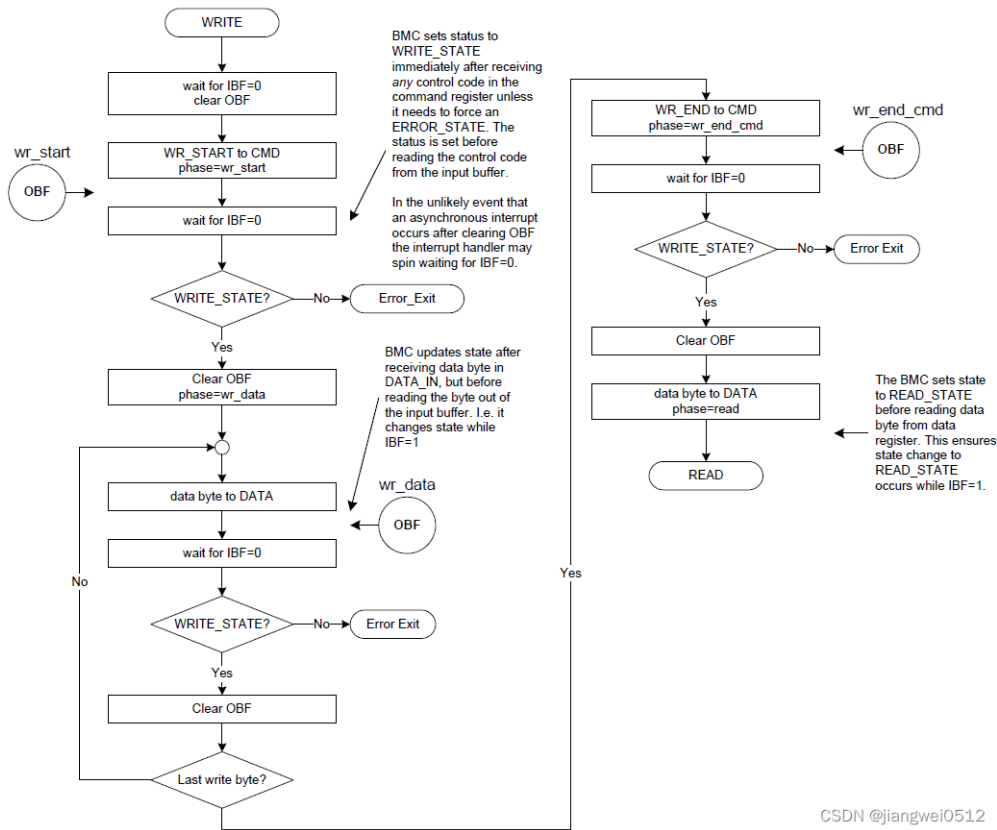Four states are obtained.

The code of Status register indicates:

```c
typedef union {
  UINT8    RawData;
  struct {
    UINT8    Obf   : 1;
    UINT8    Ibf   : 1;
    UINT8    SmAtn : 1;
    UINT8    CD    : 1;
    UINT8    Oem1  : 1;
    UINT8    Oem2  : 1;
    UINT8    State : 2;
  } Status;
} KCS_STATUS;
```

According to the above register operation, the process of BIOS writing data to BMC is obtained:

Figure 9-, KCS Interface SMS to BMC Write Transfer Flow Chart

The corresponding code implementation can be found in IpmiFeaturePkg\GenericIpmi\Common\GenericIpmi.c. The following is a brief description of the code:

1. Waiting for IBF=0:

```c
 1  do {
 2    MicroSecondDelay (IPMI_DELAY_UNIT);
 3    KcsStatus.RawData = IoRead8 (KcsIoBase + 1);
 4    if ((KcsStatus.RawData == 0xFF) || (TimeOut >= IpmiTimeoutPeriod)) {
 5      if ((Status = KcsErrorExit (IpmiTimeoutPeriod)) != EFI_SUCCESS) {
 6        return Status;
 7      }
 8    }
 9
10    TimeOut++;
11  } while (KcsStatus.Status.Ibf);
```

2. Write KCS_WRITE_START and wait for IBF=0:

```c
 1    KcsData = KCS_WRITE_START;
 2    IoWrite8 ((KcsIoBase + 1), KcsData);
 3    if ((Status = KcsCheckStatus (IpmiTimeoutPeriod, KcsWriteState, &Idle)) != EFI_SUCCESS) {
 4      return Status;
 5    }
```

3. Start writing data, wait for data writing to complete, and finally write KCS_WRITE_END :

```c
 1    for (i = 0; i < DataSize; i++) {
 2      if (i == (DataSize - 1)) {
 3        if ((Status = KcsCheckStatus (IpmiTimeoutPeriod, KcsWriteState, &Idle)) != EFI_SUCCESS) {
 4          return Status;
 5        }
 6
 7        KcsData = KCS_WRITE_END;
 8        IoWrite8 ((KcsIoBase + 1), KcsData);
 9      }
10
11      Status = KcsCheckStatus (IpmiTimeoutPeriod, KcsWriteState, &Idle);
12      if (EFI_ERROR (Status)) {
13        return Status;
14      }
15
16      IoWrite8 (KcsIoBase, Data[i]);
17    }
```

4. Start receiving data:

```c
while (TRUE) {
  if ((Status = KcsCheckStatus (IpmiTimeoutPeriod, KcsReadState, &Idle)) != EFI_SUCCESS) {
    return Status;
  }

  if (Idle) {
    *DataSize = Count;
    break;
  }

  //
  // Need to check Data Size -1 to account for array access
  //
  if (Count >= *DataSize) {
    return EFI_DEVICE_ERROR;
  }

  Data[Count] = IoRead8 (KcsIoBase);

  Count++;

  KcsData = KCS_READ;
  IoWrite8 (KcsIoBase, KcsData);
}
```

收起 ∧

5. End data reception and return the data.

The above is just a brief introduction. There are some `KcsCheckStatus()` functions that are not explained. You can directly look at the code for details.

Introduction to IPMI commands

IPMI commands are written to the KSC interface through NetFn/Cmd to complete different operations. Here are all the basic operations defined by the IPMI specification. There are several header files in the EDK code that have defined some IPMI commands. The corresponding header files are:

```c
#include <IndustryStandard/IpmiNetFnChassis.h>
#include <IndustryStandard/IpmiNetFnBridge.h>
#include <IndustryStandard/IpmiNetFnSensorEvent.h>
#include <IndustryStandard/IpmiNetFnApp.h>
#include <IndustryStandard/IpmiNetFnFirmware.h>
#include <IndustryStandard/IpmiNetFnStorage.h>
#include <IndustryStandard/IpmiNetFnTransport.h>
#include <IndustryStandard/IpmiNetFnGroupExtension.h>
#include <IndustryStandard/IpmiFruInformationStorage.h>
```

Note:

- NetFn are all even numbers.

- The O/M in the last column indicates whether the BMC needs to implement the command.

- The header file under BIOS does not contain all the commands defined in the IPMI specification, so many commands in the following table do not have corresponding macros.

- In addition to the commands defined in the IPMI specification, there are also custom commands, which is of course beneficial because it expands the content of communication between BIOS and BMC. However, different BIOS and BMC manufacturers have different implementations of the same command word, which makes maintenance difficult.

NetFn 0x00 - IPMI_NETFN_CHASSIS

The corresponding header file is edk2\MdePkg\Include\IndustryStandard\IpmiNetFnChassis.h:

Cmd	illustrate	O/M
0x00 - IPMI_CHASSIS_GET_CAPABILITIES	Get Chassis Capabilities	Required
0x01 - IPMI_CHASSIS_GET_STATUS	Get Chassis Status	Required
0x02 - IPMI_CHASSIS_CONTROL	Chassis Control	Required
0x03 - IPMI_CHASSIS_RESET	Chassis Reset	Optional
0x04 - IPMI_CHASSIS_IDENTIFY	Chassis Identify	Optional
0x05 - IPMI_CHASSIS_SET_CAPABILITIES	Set Chassis Capabilities	Optional
0x06 - IPMI_CHASSIS_SET_POWER_RESTORE_POLICY	Set Power Restore Policy	Optional
0x07 - IPMI_CHASSIS_GET_SYSTEM_RESTART_CAUSE	Get System Restart Cause	Optional
0x08 - IPMI_CHASSIS_SET_SYSTEM_BOOT_OPTIONS	Set System Boot Options	Optional
0x09 - IPMI_CHASSIS_GET_SYSTEM_BOOT_OPTIONS	Get System Boot Options	Optional
0x0A - IPMI_CHASSIS_SET_FRONT_PANEL_BUTTON_ENABLES	Set Front Panel Enables	Optional
0x0B - IPMI_CHASSIS_SET_POWER_CYCLE_INTERVALS	Set Power Cycle Interval	Optional
0x0F - IPMI_CHASSIS_GET_POH_COUNTER	Get POH Counter	Optional

NetFn 0x02 - IPMI_NETFN_BRIDGE

The corresponding header file is edk2\MdePkg\Include\IndustryStandard\IpmiNetFnBridge.h:

Cmd
0x00 - IPMI_BRIDGE_GET_STATE
0x01 - IPMI_BRIDGE_SET_STATE
0x02 - IPMI_BRIDGE_GET_ICMB_ADDRESS
0x03 - IPMI_BRIDGE_SET_ICMB_ADDRESS
0x04 - IPMI_BRIDGE_SET_PROXY_ADDRESS
0x05 - IPMI_BRIDGE_GET_BRIDGE_STATISTICS
0x06 - IPMI_BRIDGE_GET_ICMB_CAPABILITIES
0x08 - IPMI_BRIDGE_CLEAR_STATISTICS
0x09 - IPMI_BRIDGE_GET_PROXY_ADDRESS
0x0A - IPMI_BRIDGE_GET_ICMB_CONNECTOR_INFO
0x0B - IPMI_BRIDGE_GET_ICMB_CONNECTION_ID
0x0C - IPMI_BRIDGE_SEND_ICMB_CONNECTION_ID
0x10 - IPMI_BRIDGE_PREPARE_FOR_DISCOVERY
0x11 - IPMI_BRIDGE_GET_ADDRESSES
0x12 - IPMI_BRIDGE_SET_DISCOVERED
0x13 - IPMI_BRIDGE_GET_CHASSIS_DEVICEID
0x14 - IPMI_BRIDGE_SET_CHASSIS_DEVICEID
0x20 - IPMI_BRIDGE_REQUEST
0x21 - IPMI_BRIDGE_MESSAGE
0x30 - IPMI_BRIDGE_GET_EVENT_COUNT
0x31 - IPMI_BRIDGE_SET_EVENT_DESTINATION
0x32 - IPMI_BRIDGE_SET_EVENT_RECEPTION_STATE
0x33 - IPMI_BRIDGE_SEND_ICMB_EVENT_MESSAGE

But they are not in the IPMI specification, but in the ICMB specification.

NetFn 0x04 - IPMI_NETFN_SENSOR_EVENT

The corresponding header file is edk2\MdePkg\Include\IndustryStandard\IpmiNetFnSensorEvent.h, but there is only one of them, and the others are blank and do not appear in the code:

Cmd	illustrate	O/M
0x00	Set Event Receiver	Required
0x01	Get Event Receiver	Required
0x02 - IPMI_SENSOR_PLATFORM_EVENT_MESSAGE	Platform Event (aka Event Message)	Required
0x10	Get PEF Capabilities	Required
0x11	Arm PEF Postpone Timer	Required
0x12	Set PEF Configuration Parameters	Required
0x13	Get PEF Configuration Parameters	Required
0x14	Set Last Processed Event ID	Required
0x15	Get Last Processed Event ID	Required
0x16	Alert Immediate	Optional
0x17	PET Acknowledge	Optional
0x20	Get Device SDR Info	Optional
0x21	Get Device SDR	Optional
0x22	Reserve Device SDR Repository	Optional
0x23	Get Sensor Reading Factors	Optional
0x24	Set Sensor Hysteresis	Optional
0x25	Get Sensor Hysteresis	Optional
0x26	Set Sensor Threshold	Optional
0x27	Get Sensor Threshold	Optional
0x28	Set Sensor Event Enable	Optional
0x29	Get Sensor Event Enable	Optional
0x2A	Re-arm Sensor Events	Optional
0x2B	Get Sensor Event Status	Optional
0x2D	Get Sensor Reading	Required

Cmd	illustrate	O/M
0x2E	Set Sensor Type	Optional
0x2F	Get Sensor Type	Optional
0x30	Set Sensor Reading And Event Status	Optional

NetFn 0x06 - IPMI_NETFN_APP

Corresponding to the header file edk2\MdePkg\Include\IndustryStandard\IpmiNetFnApp.h, some blanks are not included in the code:

Cmd	illustrate	O/M
0x01 - IPMI_APP_GET_DEVICE_ID	Get Device ID	Required
0x02 - IPMI_APP_COLD_RESET	Cold Rest	Optional
0x03 - IPMI_APP_WARM_RESET	Warm Rest	Optional
0x04 - IPMI_APP_GET_SELFTEST_RESULTS	Get Selft Test Results	Required
0x05 - IPMI_APP_MANUFACTURING_TEST_ON	Manufacturing Test On	Optional
0x06 - IPMI_APP_SET_ACPI_POWERSTATE	Set ACPI Power State	Optional
0x07 - IPMI_APP_GET_ACPI_POWERSTATE	Get ACPI Power State	Optional
0x08 - IPMI_APP_GET_DEVICE_GUID	Get Device GUID	Optional
0x09	Get NetFun Support	Optional
0x0A	Get Command Support	Optional
0x0B	Get Command Sub-function Support	Optional
0x0C	Get Configurable Commands	Optional
0x0D	Get Configurable Command Sub-functions	Optional
0x22 - IPMI_APP_RESET_WATCHDOG_TIMER	Reset Watchdog Timer	Required
0x24 - IPMI_APP_SET_WATCHDOG_TIMER	Set Watchdog Timer	Required
0x25 - IPMI_APP_GET_WATCHDOG_TIMER	Get Watchdog Timer	Required
0x2E - IPMI_APP_SET_BMC_GLOBAL_ENABLES	Set BMC Global Enables	Required
0x2F - IPMI_APP_GET_BMC_GLOBAL_ENABLES	Get BMC Global Enables	Required
0x30 - IPMI_APP_CLEAR_MESSAGE_FLAGS	Clear Message Flags	Required
0x31 - IPMI_APP_GET_MESSAGE_FLAGS	Get Message Flags	Required
0x32 - IPMI_APP_ENABLE_MESSAGE_CHANNEL_RECEIVE	Enable Message Channel Receive	Optional
0x33 - IPMI_APP_GET_MESSAGE	Get Message	Required
0x34 - IPMI_APP_SEND_MESSAGE	Send Message	Required
0x35 - IPMI_APP_READ_EVENT_MSG_BUFFER	Read Event Message Buffer	Optional
0x36 - IPMI_APP_GET_BT_INTERFACE_CAPABILITY	Get BT Interface Capabilities	Required
0x37 - IPMI_APP_GET_SYSTEM_GUID	Get System GUID	Optional
0x38 - IPMI_APP_GET_CHANNEL_AUTHENTICATION_CAPABILITIES	Get Channel Authentication Capabilities	Optional
0x39 - IPMI_APP_GET_SESSION_CHALLENGE	Get Session Challenge	Optional
0x3A - IPMI_APP_ACTIVATE_SESSION	Activate Session	Optional
0x3B - IPMI_APP_SET_SESSION_PRIVELEGE_LEVEL	Set Session Privilege Level	Optional
0x3C - IPMI_APP_CLOSE_SESSION	Close Session	Optional
0x3D - IPMI_APP_GET_SESSION_INFO	Get Session Info	Optional
0x3F - IPMI_APP_GET_AUTHCODE	Get AuthCode	Optional
0x40 - IPMI_APP_SET_CHANNEL_ACCESS	Set Channel Access	Optional
0x41 - IPMI_APP_GET_CHANNEL_ACCESS	Get Channel Access	Optional
0x42 - IPMI_APP_GET_CHANNEL_INFO	Get Channel Info	Optional
0x43 - IPMI_APP_SET_USER_ACCESS	Set User Access	Optional
0x44 - IPMI_APP_GET_USER_ACCESS	Get User Access	Optional
0x45 - IPMI_APP_SET_USER_NAME	Set User Name	Optional
0x46 - IPMI_APP_GET_USER_NAME	Get User Name	Optional
0x47 - IPMI_APP_SET_USER_PASSWORD	Set User Password	Optional
0x48 - IPMI_APP_ACTIVATE_PAYLOAD	Activate Payload	Optional
0x49 - IPMI_APP_DEACTIVATE_PAYLOAD	Deactivate Payload	Optional
0x4A - IPMI_APP_GET_PAYLOAD_ACTIVATION_STATUS	Get Payload Activation Status	Optional
0x4B - IPMI_APP_GET_PAYLOAD_INSTANCE_INFO	Get Payload Instance Info	Optional

Cmd	illustrate	O/M
0x4C - IPMI_APP_SET_USER_PAYLOAD_ACCESS	Set User Payload Access Command	Optional
0x4D - IPMI_APP_GET_USER_PAYLOAD_ACCESS	Get User Payload Access Command	Optional
0x4E - IPMI_APP_GET_CHANNEL_PAYLOAD_SUPPORT	Get Channel Payload Support Command	Optional
0x4F - IPMI_APP_GET_CHANNEL_PAYLOAD_VERSION	Get Channel Payload Version Command	Optional
0x50 - IPMI_APP_GET_CHANNEL_OEM_PAYLOAD_INFO	Get Channel OEM Payload Info Command	Optional
0x52 - IPMI_APP_MASTER_WRITE_READ	Master Write-Read	Required
0x54 - IPMI_APP_GET_CHANNEL_CIPHER_SUITES	Get Channel Cipher Suites	Optional
0x55 - IPMI_APP_SUSPEND_RESUME_PAYLOAD_ENCRYPTION	Suspend/Resume Payload Encryption Command	Optional
0x56 - IPMI_APP_SET_CHANNEL_SECURITY_KEYS	Set Channel Security Keys	Optional
0x57 - IPMI_APP_GET_SYSTEM_INTERFACE_CAPABILITIES	Get System Interface Capabilities	Optional
0x58	Set System Info Parameters	Optional
0x59	Get System Info Parameters	Optional
0x60	Set Command Enables	Optional
0x61	Get Command Enables	Optional
0x62	Set Command Sub-function Enables	Optional
0x63	Get Command Sub-function Enables	Optional
0x64	Get OEM NetFn IANA Support	Optional

IPMI_APP_GET_DEVICE_ID also has a broadcast version Broadcast "Get Deivce ID", and its Cmd is also 0x01.

NetFn 0x08 - IPMI_NETFN_FIRMWARE

The corresponding header file is edk2\MdePkg\Include\IndustryStandard\IpmiNetFnFirmware.h, but the following commands are not found in the IPMI specification:

Cmd
0x23 - IPMI_GET_BMC_EXECUTION_CONTEXT

NetFn 0x0A - IPMI_NETFN_STORAGE

The corresponding header file is edk2\MdePkg\Include\IndustryStandard\IpmiNetFnStorage.h:

Cmd	illustrate	O/M
0x10 - IPMI_STORAGE_GET_FRU_INVENTORY_AREAINFO	Get FRU Inventory Area Info	Required
0x11 - IPMI_STORAGE_READ_FRU_DATA	Read FRU Data	Required
0x12 - IPMI_STORAGE_WRITE_FRU_DATA	Write FRU Data	Required
0x20 - IPMI_STORAGE_GET_SDR_REPOSITORY_INFO	Get SDR Repository Info	Required
0x21 - IPMI_STORAGE_GET_SDR_REPOSITORY_ALLOCATION_INFO	Get SDR Pepository Allocation Info	Optional
0x22 - IPMI_STORAGE_RESERVE_SDR_REPOSITORY	Reserve SDR Repository	Required
0x23 - IPMI_STORAGE_GET_SDR	Get SDR	Required
0x24 - IPMI_STORAGE_ADD_SDR	Add SDR	Required
0x25 - IPMI_STORAGE_PARTIAL_ADD_SDR	Partial Add SDR	Required
0x26 - IPMI_STORAGE_DELETE_SDR	Delete SDR	Optional
0x27 - IPMI_STORAGE_CLEAR_SDR	Clear SDR Repository	Required
0x28 - IPMI_STORAGE_GET_SDR_REPOSITORY_TIME	Get SDR Repository Time	Optional
0x29 - IPMI_STORAGE_SET_SDR_REPOSITORY_TIME	Set SDR Repository Time	Optional
0x2A - IPMI_STORAGE_ENTER_SDR_UPDATE_MODE	Enter SDR Pepository Update Mode	Optional
0x2B - IPMI_STORAGE_EXIT_SDR_UPDATE_MODE	Exit SDR Repository Update Mode	Optional
0x2C - IPMI_STORAGE_RUN_INIT_AGENT	Run Initialization Agent	Optional
0x40 - IPMI_STORAGE_GET_SEL_INFO	Get SEL Info	Required
0x41 - IPMI_STORAGE_GET_SEL_ALLOCATION_INFO	Get SEL Allocation Info	Optional
0x42 - IPMI_STORAGE_RESERVE_SEL	Reserve SEL	Optional
0x43 - IPMI_STORAGE_GET_SEL_ENTRY	Get SEL Entry	Required
0x44 - IPMI_STORAGE_ADD_SEL_ENTRY	Add SEL Entry	Required
0x45 - IPMI_STORAGE_PARTIAL_ADD_SEL_ENTRY	Partial Add SEL Entry	Required
0x46 - IPMI_STORAGE_DELETE_SEL_ENTRY	Delete SEL Entry	Optional
0x47 - IPMI_STORAGE_CLEAR_SEL	Clear SEL	Required
0x48 - IPMI_STORAGE_GET_SEL_TIME	Get SEL Time	Required
0x49 - IPMI_STORAGE_SET_SEL_TIME	Set SEL Time	Required

Cmd	illustrate	O/M
0x5A - IPMI_STORAGE_GET_AUXILLARY_LOG_STATUS	Get Auxiliary Log Status	Optional
0x5B - IPMI_STORAGE_SET_AUXILLARY_LOG_STATUS	Set Auxiliary Log Status	Optional
0x5C - IPMI_STORAGE_GET_SEL_TIME_UTC_OFFSET	Get SEL Timer UTC Offset	Optional
0x5D - IPMI_STORAGE_SET_SEL_TIME_UTC_OFFSET	Set SEL Timer UTC Offset	Optional

NetFn 0x0C - IPMI_NETFN_TRANSPORT

Corresponding to the header file edk2\MdePkg\Include\IndustryStandard\IpmiNetFnTransport.h, some blanks are not in the code:

Cmd	illustrate	O/M
0x01 - IPMI_TRANSPORT_SET_LAN_CONFIG_PARAMETERS	Set LAN Configuration Parameters	Required
0x02 - IPMI_TRANSPORT_GET_LAN_CONFIG_PARAMETERS	Get LAN Configuration Parameters	Required
0x03 - IPMI_TRANSPORT_SUSPEND_BMC_ARPS	Suspend BMC ARPs	Optional
0x04 - IPMI_TRANSPORT_GET_PACKET_STATISTICS	Get IP/UDP/RMCP Statistics	Optional
0x10 - IPMI_TRANSPORT_SET_SERIAL_CONFIGURATION	Set Serial/Modem Configuration	Required
0x11 - IPMI_TRANSPORT_GET_SERIAL_CONFIGURATION	Get Serial/Modem Configuration	Required
0x12 - IPMI_TRANSPORT_SET_SERIAL_MUX	Set Serial/Modem Mux	Optional
0x13 - IPMI_TRANSPORT_GET_TAP_RESPONSE_CODE	Get TAP Response Codes	Optional
0x14 - IPMI_TRANSPORT_SET_PPP_UDP_PROXY_TXDATA	Set PPP UDP Proxy Transmit Data	Optional
0x15 - IPMI_TRANSPORT_GET_PPP_UDP_PROXY_TXDATA	Get PPP UDP Proxy Transmit Data	Optional
0x16 - IPMI_TRANSPORT_SEND_PPP_UDP_PROXY_PACKET	Send PPP UDP Proxy Packet	Optional
0x17 - IPMI_TRANSPORT_GET_PPP_UDP_PROXY_RX	Get PPP UDP Proxy Receive Data	Optional
0x18 - IPMI_TRANSPORT_SERIAL_CONNECTION_ACTIVE	Serial/Modem Connection Active	Required
0x19 - IPMI_TRANSPORT_CALLBACK	Callback	Optional
0x1A - IPMI_TRANSPORT_SET_USER_CALLBACK_OPTIONS	Set User Callback Options	Optional
0x1B - IPMI_TRANSPORT_GET_USER_CALLBACK_OPTIONS	Get User Callback Options	Optional
0x1C	Set Serial Routing Mux	Optional
0x20 - IPMI_TRANSPORT_SOL_ACTIVATING	SOL Activating	Optional
0x21 - IPMI_TRANSPORT_SET_SOL_CONFIG_PARAM	Set SOL Configuration Parameters	Optional
0x22 - IPMI_TRANSPORT_GET_SOL_CONFIG_PARAM	Get SOL Configuration Parameters	Optional
0x30	Forwarded Command	Optional
0x31	Set Forwarded Commands	Optional
0x32	Get Forwarded Commands	Optional
0x33	Enable Forwarded Commands	Optional

NetFn 0x2C - IPMI_NETFN_GROUP_EXT

The corresponding header file is edk2\MdePkg\Include\IndustryStandard\IpmiNetFnGroupExtension.h, but there is no specific content in it. It is used for extension. This actually leads to additional problems, because the commands provided by different BIOS vendors may be different, making maintenance difficult.