Examining TPM2 ACPI Table

Finnbarr P. Murphy

(fpm@fpmurphy.com)

The Advanced Configuration and Power Interface (ACPI) specification was developed to establish industry common interfaces enabling robust operating system directed motherboard device configuration and power management of both devices and entire platforms. This specification has gone from strength to strength over the years and is now maintained by the UEFI Forum. The current version is 6.1.

Over the years, the Trusted Computing Group (TCG) has developed various specifications defining an ACPI table and basic methods for use on a TCG compliant platform. The goal is that the ACPI table and ACPI namespace objects provide sufficient information to an operating system to enable access to the TCG compliant hardware in a platform.

The current TCG ACPI Specification is the *Family "1.2" and "2.0" Level 00 Revision 00.37* dated December 19, 2014. This specification defines separate ACPI tables for 1.2 TPM and 2.0 TPM hardware. In this post, I examine the TPM2 ACPI table.

All TCG platforms supporting ACPI utilize the same header section layout. Table 2 describes the client ACPI table for TPM 1.2. Table 4 of the specification describes the server ACPI table for TPM 1.2. Table 7 describes the ACPI table for TPM 2.0, which is used for client or server platforms. The value in the signature field for the TPM 1.2 tables ('TCPA') is different from the value for the TPM 2.0 table ('TPM2').

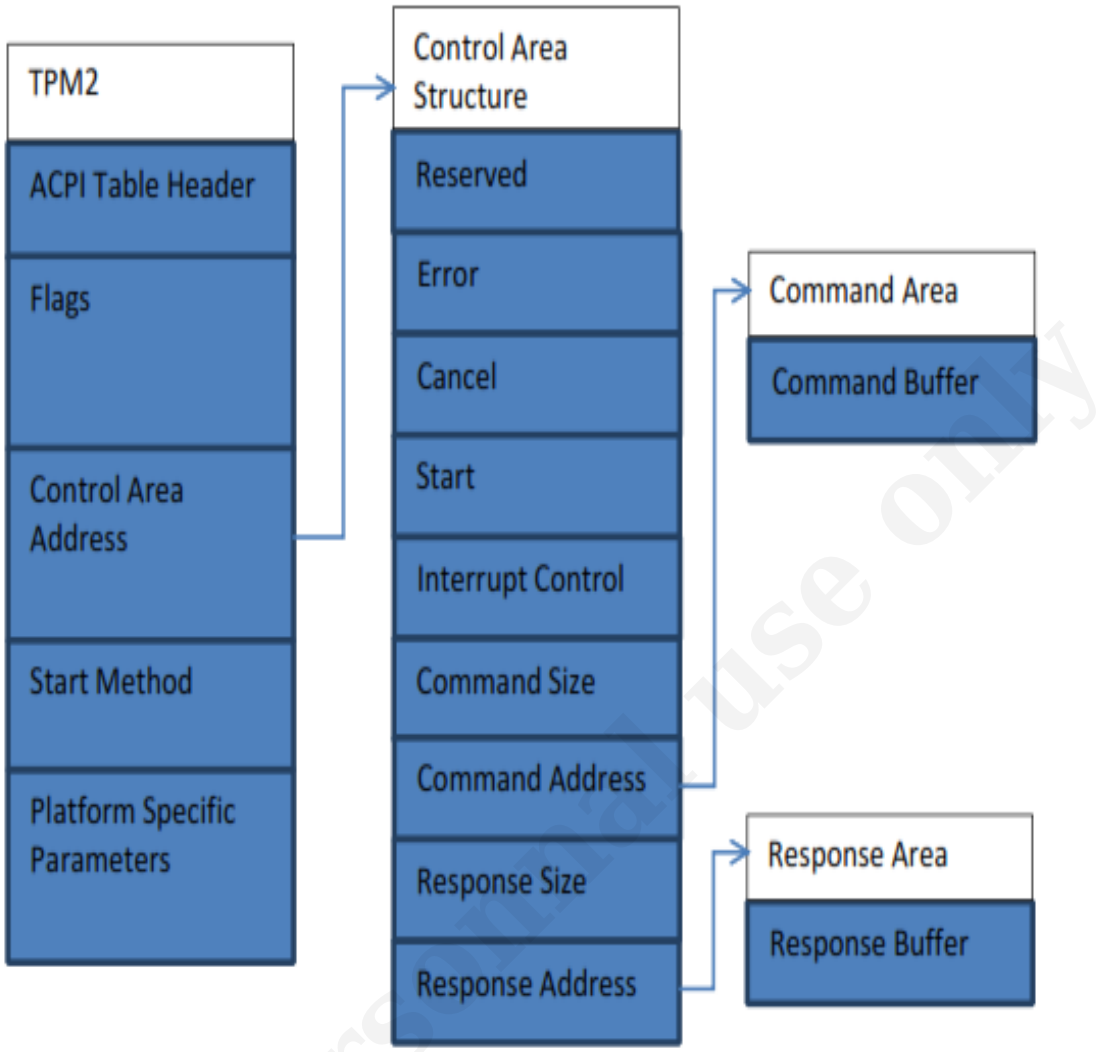
Prior to the development of the current TCG TPM ACPI specification, Microsoft published a document called *The Microsoft TPM 2.0 Hardware Interface Table* specification in November 2011 This is now marked obsolete by Microsoft. Incidentally, on their webpage referring to this document, Microsoft refers to the TCG as the *Thrustworthy Computing Group*! In this document, it is recommended that TPM 2.0 device object ACPI table appears under the RDST table rather than the DSDT table in the ACPI namespace.

Here is the relevant section from the TCG ACPI Specification, Family "1.2" and "2.0", Level 00 Revision 00.37 December 19, 2014 for the TPM2 table:

Table 7: TCG Hardware Interface Description Table Format for TPM 2.0

Field	Byte Length	Byte Offset	Description
Header			
Signature	4	0	'TPM2'. Signature for the TCG Hardware Interface Table.
Length	4	4	The length of this table starting from the Signature field up to and including the platform specific parameters field. (52 + size of platform specific parameters)
Revision	1	8	4. The current revision of this table.
Checksum	1	9	Entire table must sum to zero.
OEMID	6	10	OEM ID. Per ACPI specification. An OEM-supplied string that identifies the OEM.
OEM Table ID	8	16	For the TPM Interface Table, the table ID is the manufacturer model ID (assigned by the OEM identified by "OEM ID").
OEM Revision	4	24	OEM revision of TPM Interface Table for the given OEM Table ID. Per ACPI, this is "An OEM-supplied revision number. Larger numbers are assumed to be newer revisions."
Creator ID	4	28	Vendor ID of utility that created the table. For the tables containing Definition Blocks, this is the ID for the ASL Compiler.
Creator Revision	4	32	Revision of utility that created the table. For the tables containing Definition Blocks, this is the revision for the ASL Compiler.
Platform Class	2	36	0 for client platforms. 1 for server platforms.
Reserved	2	38	0
Address of Control Area	8	40	Physical address of Control Area. The Control Area contains status registers and the location of the memory buffers for communicating with the device. The area may be in either TPM 2.0 device memory or in memory reserved by the system during boot. Interfaces that do not require the Control Area set this value to zero.
Start Method	4	48	The Start Method selector determines which mechanism the device driver uses to notify the TPM 2.0 device that a command is available for processing. This field may contain one of the values specified in Table 8.
Platform Specific Parameters	Variable	52	The content of the platform specific parameters is determined by the Start Method used by the system's TPM device interface. This field contains values that may be used to initiate command processing. This information may be vendor specific. If the Start Method value is 2 then this field is four bytes in size and must be all zero.

The following diagram comes from the Microsoft document entitled *TPM 2.0 Hardware Interface Table (TPM2)* which we discussed earlier. It is useful in that it shows the interconnection between the TCP2 ACPI table and the underlying TPM2 firmware control structures.



Note that the *ControlArea* structure is not part of the ACPI TPM2 Table. It is a platform and OS implementation detail. Typically the structure contains input *command) and output (response) buffers, status fields and more. Software will write TPM commands to be executed by a TPM device to the command buffer and read responses from the TPM device in the response buffer. However all of this is vendor and device specific.

Here is the source code for a simple UEFI utility which will print out the details of the ACPI TPM2 table from the UEFI shell:

```
//
//  Copyright (c) 2016  Finnbarr P. Murphy.   All rights reserved.
//
//  Show ACPI TPM2 table details
//
//  License: BSD License
//
#include <Uefi.h>
#include <Library/UefiLib.h>
#include <Library/ShellCEntryLib.h>
#include <Library/ShellLib.h>
#include <Library/BaseLib.h>
```



```
#include <Library/BaseMemoryLib.h>
#include <Library/MemoryAllocationLib.h>
#include <Library/UefiBootServicesTableLib.h>
#include <Library/PrintLib.h>
#include <Protocol/EfiShell.h>
#include <Protocol/LoadedImage.h>
#include <Protocol/AcpiSystemDescriptionTable.h>
#pragma pack (1)
typedef struct _MY_EFI_TPM2_ACPI_TABLE {
    EFI_ACPI_SDT_HEADER  Header;
    UINT16               PlatformClass;
    UINT16               Reserved;
    UINT64               ControlAreaAddress;
    UINT32               StartMethod;
 // UINT8                PlatformSpecificParameters[];
} MY_EFI_TPM2_ACPI_TABLE;
#pragma pack()
#define EFI_ACPI_20_TABLE_GUID \
    { 0x8868e871, 0xe4f1, 0x11d3, {0xbc, 0x22, 0x0, 0x80, 0xc7, 0x3c, 0x88, 0x81 }}
int Verbose = 0;
static VOID
AsciiToUnicodeSize( CHAR8 *String,
                    UINT8 length,
                    CHAR16 *UniString)
{
    int len = length;
    while (*String != '&#92;&#48;' &amp;&amp; len > 0) {
        *(UniString++) = (CHAR16) *(String++);
        len--;
    }
    *UniString = '&#92;&#48;';
}
//
// Print Start Method details
//
static VOID
PrintStartMethod( UINT32 StartMethod)
{
    Print(L"        Start Method : %d (", StartMethod);
    switch (StartMethod) {
        case 0:  Print(L"Not allowed");
                 break;
        case 1:  Print(L"vnedor specific legacy use");
                 break;
        case 2:  Print(L"ACPI start method");
                 break;
        case 3:
        case 4:
        case 5:  Print(L"Verndor specific legacy use");
                 break;
        case 6:  Print(L"Memory mapped I/O");
                 break;
        case 7:  Print(L"Command response buffer interface");
                 break;
        case 8:  Print(L"Command response buffer interface, ACPI start method");
                 break;
        default: Print(L"Reserved for future use");
    }
    Print(L")\n");
}
//
// Parse and print TCM2 Table details
//
static VOID
ParseTPM2(MY_EFI_TPM2_ACPI_TABLE *Tpm2)
{
    CHAR16 Buffer[100];
    UINT8  PlatformSpecificMethodsSize = Tpm2->Header.Length - 52;
```



```
    Print(L"\n");
    AsciiToUnicodeSize((CHAR8 *)&amp;(Tpm2->Header.Signature), 4, Buffer);
    Print(L"           Signature : %s\n", Buffer);
    Print(L"              Length : %d\n", Tpm2->Header.Length);
    Print(L"            Revision : %d\n", Tpm2->Header.Revision);
    Print(L"            Checksum : %d\n", Tpm2->Header.Checksum);
    AsciiToUnicodeSize((CHAR8 *)(Tpm2->Header.OemId), 6, Buffer);
    Print(L"              Oem ID : %s\n", Buffer);
    AsciiToUnicodeSize((CHAR8 *)(Tpm2->Header.OemTableId), 8, Buffer);
    Print(L"        Oem Table ID : %s\n", Buffer);
    Print(L"        Oem Revision : %d\n", Tpm2->Header.OemRevision);
    AsciiToUnicodeSize((CHAR8 *)&amp;(Tpm2->Header.CreatorId), 4, Buffer);
    Print(L"          Creator ID : %s\n", Buffer);
    Print(L"    Creator Revision : %d\n", Tpm2->Header.CreatorRevision);
    Print(L"      Platform Class : %d\n", Tpm2->PlatformClass);
    Print(L"Control Area Address : %lld\n", Tpm2->ControlAreaAddress);
    PrintStartMethod(Tpm2->StartMethod);
    Print(L"  Platform S.P. Size : %d\n", PlatformSpecificMethodsSize);
    Print(L"\n");
}
static int
ParseRSDP( EFI_ACPI_2_0_ROOT_SYSTEM_DESCRIPTION_POINTER *Rsdp,
           CHAR16* GuidStr)
{
    EFI_ACPI_SDT_HEADER *Xsdt, *Entry;
    CHAR16 OemStr[20];
    UINT32 EntryCount;
    UINT64 *EntryPtr;
    AsciiToUnicodeSize((CHAR8 *)(Rsdp->OemId), 6, OemStr);
    if (Rsdp->Revision >= EFI_ACPI_2_0_ROOT_SYSTEM_DESCRIPTION_POINTER_REVISION) {
        Xsdt = (EFI_ACPI_SDT_HEADER *)(Rsdp->XsdtAddress);
    } else {
    if (Verbose) {
        Print(L"ERROR: Invalid RSDP revision number.\n");
    }
        return 1;
    }
    if (Xsdt->Signature != SIGNATURE_32 ('X', 'S', 'D', 'T')) {
    if (Verbose) {
        Print(L"ERROR: XSDT table signature not found.\n");
    }
        return 1;
    }
    AsciiToUnicodeSize((CHAR8 *)(Xsdt->OemId), 6, OemStr);
    EntryCount = (Xsdt->Length - sizeof (EFI_ACPI_SDT_HEADER)) / sizeof(UINT64);
    EntryPtr = (UINT64 *)(Xsdt + 1);
    for (int Index = 0; Index < EntryCount; Index++, EntryPtr++) {
        Entry = (EFI_ACPI_SDT_HEADER *)((UINTN)(*EntryPtr));
        if (Entry->Signature == SIGNATURE_32 ('T', 'P', 'M', '2')) {
            ParseTPM2((MY_EFI_TPM2_ACPI_TABLE *)((UINTN)(*EntryPtr)));
        }
    }
    return 0;
}
static void
Usage(void)
{
    Print(L"Usage: ShowTPM2 [-v|--verbose]\n");
}
INTN
EFIAPI
ShellAppMain(UINTN Argc, CHAR16 **Argv)
{
    EFI_CONFIGURATION_TABLE *ect = gST->ConfigurationTable;
    EFI_ACPI_2_0_ROOT_SYSTEM_DESCRIPTION_POINTER *Rsdp = NULL;
    EFI_GUID Acpi20TableGuid = EFI_ACPI_20_TABLE_GUID;
    EFI_STATUS Status = EFI_SUCCESS;
    CHAR16 GuidStr[100];
```



```
    for (int i = 1; i < Argc; i++) {
        if (!StrCmp(Argv[i], L"--verbose") ||
            !StrCmp(Argv[i], L"-v")) {
            Verbose = 1;
        } else if (!StrCmp(Argv[i], L"--help") ||
            !StrCmp(Argv[i], L"-h") ||
            !StrCmp(Argv[i], L"-?")) {
            Usage();
            return Status;
        } else {
            Print(L"ERROR: Unknown option.\n");
            Usage();
            return Status;
        }
    }
    // locate RSDP (Root System Description Pointer)
    for (int i = 0; i < gST->NumberOfTableEntries; i++) {
    if (CompareGuid (&amp;(gST->ConfigurationTable[i].VendorGuid), &amp;Acpi20TableGuid))
{
        if (!AsciiStrnCmp("RSD PTR ", (CHAR8 *)(ect->VendorTable), 8)) {
        UnicodeSPrint(GuidStr, sizeof(GuidStr), L"%g", &amp;(gST->ConfigurationTable[i].Ve
ndorGuid));
        Rsdp = (EFI_ACPI_2_0_ROOT_SYSTEM_DESCRIPTION_POINTER *)ect->VendorTable;
            ParseRSDP(Rsdp, GuidStr);
        }
    }
    ect++;
    }
    if (Rsdp == NULL) {
    if (Verbose) {
        Print(L"ERROR: Could not find an ACPI RSDP table.\n");
    }
    return EFI_NOT_FOUND;
    }
    return Status;
}
```

This utility will build within UDK2015 using the following .INF file:

```
[Defines]
  INF_VERSION                    = 0x00010006
  BASE_NAME                      = ShowTPM2
  FILE_GUID                      = 4ea87c51-7491-4dfd-0055-747010f3ce51
  MODULE_TYPE                    = UEFI_APPLICATION
  VERSION_STRING                 = 0.1
  ENTRY_POINT                    = ShellCEntryLib
  VALID_ARCHITECTURES            = X64

[Sources]
  ShowTPM2.c

[Packages]
  MdePkg/MdePkg.dec
  ShellPkg/ShellPkg.dec


[LibraryClasses]
  ShellCEntryLib
  ShellLib
  BaseLib
  BaseMemoryLib
  UefiLib

[Protocols]
```



```
[BuildOptions]

[Pcd]
```

These two files are available in my *UEFI-Utilities-2016* repository at GitHub.

Here is the output I get on my Lenovo T450 when I run the utility:

```
fs0> ShowTPM2 -h
Usage: ShowTPM2 [-v|--verbose]

fs0> ShowTPM2

           Signature : TPM2
              Length : 52
            Revision : 3
            Checksum : 57
              Oem ID : LENOVO
        Oem Table ID : TP-JB
        Oem Revision : 4608
          Creator ID : PTEC
    Creator Revision : 2
      Platform Class : 0
Control Area Address : 3437228032
        Start Method : 2 (ACPI start method)
  Platform S.P. Size : 0

fs0>
```

Note that the last line, i.e. Platform S.P. Size, has a value of 0. According to the TCG specification referenced above, the Platform Specific Parameters Size field should have been 4 bytes in size and all zeros for a type 2 Start Method.

I am not sure what the creator ID of *PTEC* actually refers too, but I suspect it refers to Phoenix Technologies.

Using the publicly available *acpidump* utility, I confirmed that indeed these 4 zero bytes are missing on my Lenovo T450 test platform. See below.

```
fs0:> acpidump.efi -e TCP2

TPM2 @ 0x00000000CCDD2000
  0000: 54 50 4D 32 34 00 00 00 03 39 4C 45 4E 4F 56 4F  TPM24....9LENOVO
  0010: 54 50 2D 4A 42 20 20 20 00 12 00 00 50 54 45 43  TP-JB   ....PTEC
  0020: 02 00 00 00 00 00 00 00 00 F0 DF CC 00 00 00 00  ................
  0030: 02 00 00 00
```

Digging deeper into the EDK2 source code revealed that, for some undocumented reason or another, the authors of the relevant code simply ignored the Platform Specific Parameters field of the ACPI TPM2 table as shown below:

```
$ vi ./SecurityPkg/Tcg/TrEESmm/TrEESmm.c

EFI_TPM2_ACPI_TABLE  mTpm2AcpiTemplate = {
  {
    EFI_ACPI_5_0_TRUSTED_COMPUTING_PLATFORM_2_TABLE_SIGNATURE,
    sizeof (mTpm2AcpiTemplate),
    EFI_TPM2_ACPI_TABLE_REVISION,
```



```
    //
    // Compiler initializes the remaining bytes to 0
    // These fields should be filled in in production
    //
  },
  0, // Flags
  0, // Control Area
  EFI_TPM2_ACPI_TABLE_START_METHOD_TIS, // StartMethod
};

$ vi ./MdePkg/Include/IndustryStandard/Tpm2Acpi.h

#ifndef _TPM2_ACPI_H_
#define _TPM2_ACPI_H_

#include

#pragma pack (1)

#define EFI_TPM2_ACPI_TABLE_REVISION  3

pragma pack (1)
typedef struct {
  EFI_ACPI_DESCRIPTION_HEADER Header;
  UINT32                      Flags;
  UINT64                      AddressOfControlArea;
  UINT32                      StartMethod;
//UINT8                       PlatformSpecificParameters[];
} EFI_TPM2_ACPI_TABLE;
#pragma pack()
```

I have no idea why this is the case but it does “explain” the Lenovo TPM2 table output.

Turning to another EDK2 issue, i.e. that of ACPI table description headers. EDK2 code is interesting in that it includes a number of typedefs for this particular header. For example consider the following two definitions.

```


pragma pack(1)

typedef struct {
  UINT32  Signature;
  UINT32  Length;
  UINT8   Revision;
  UINT8   Checksum;
  UINT8   OemId[6];
  UINT64  OemTableId;
  UINT32  OemRevision;
  UINT32  CreatorId;
  UINT32  CreatorRevision;
} EFI_ACPI_DESCRIPTION_HEADER;

typedef struct {
  UINT32    Signature;
  UINT32    Length;
  UINT8     Revision;
  UINT8     Checksum;
  CHAR8     OemId[6];
  CHAR8     OemTableId[8];
  UINT32    OemRevision;
  UINT32    CreatorId;
  UINT32    CreatorRevision;
} EFI_ACPI_SDT_HEADER;
#pragma pack()
```


Why two definitions for the same header?

The current ACPI standard defines the table description header as follows:

```
DefBlockHeader := TableSignature TableLength SpecCompliance CheckSum OemID
                  OemTableID OemRevision CreatorID CreatorRevision

TableSignature := DWordData    // As defined in section 5.2.3.
TableLength := DWordData       // Length of the table in bytes including the block header.
SpecCompliance := ByteData     // The revision of the structure.
CheckSum := ByteData           // Byte checksum of the entire table.
OemID := ByteData(6)           // OEM ID of up to 6 characters.
OemTableID := ByteData(8)      // OEM Table ID of up to 8 characters.
OemRevision := DWordData       // OEM Table Revision.
CreatorID := DWordData         // Vendor ID of the ASL compiler.
CreatorRevision := DWordData   // Revision of the ASL compiler.
```

I believe that the second definition is closer to the intent of the ACPI

For a more detailed look at the actual TPM2 support in the EDK2, read the Intel white paper entitled *A Tour Beyond BIOS with the UEFI TPM2 Support in EDKII* by Jiewen Yao and Vincent J. Zimmer.

