Retrieve Microsoft Windows Product Key From UEFI Shell

Finnbarr P. Murphy

(fpm@fpmurphy.com)

With Microsoft Windows v3.0 activation, the product key is now stored in the UEFI firmware instead of being printed on a COA sticker. Specifically, on UEFI firmware, the individualized product key is stored in the ACPI (Advanced Configuration and Power Interface) table called MSDM (Microsoft Data Management). This table contains the information necessary to enable individualized OEM activation.

Field	Byte Length	Byte Offset	Description
Signature	4	0	MSDM
Length	4	4	Length, in bytes, of the entire table.
Revision	1	8	0x01
Checksum	1	9	Checksum of the entire table.
OEMID	6	10	An OEM-supplied string that identifies the OEM.
OEM Table ID	8	16	Optional motherboard/BIOS logical identifier.
OEM Revision	4	24	OEM revision number of the table for the supplied OEM Table ID.
Creator ID	4	28	Vendor ID of the utility that created the table.
Creator Revision	4	32	Revision of the utility that created the table.
Software Licensing Structure	Variable length	36	Proprietary data structure that contains all the licensing data necessary to enable Windows activation. Details can be found in the appropriate Microsoft OEM licensing kit by first visiting the Microsoft OEM website

Note that generic activation information is stored in a separate ACPI table named SLIC (Software Licensing). I will not be discussing that particular table in this post.

This utility is quite simple. It locates a pointer to the list of ACPI tables, looks for a table with a MSDM signature in the table header and, if found, prints out either the full table information or just the product key.

Here is the source code for the utility:

```
//
//  Copyright (c) 2015  Finnbarr P. Murphy.   All rights reserved.
//
//  Display MSDM table information including Windows product key
//
//  License: BSD License
//
#include <efi.h>
#include <efilib.h>
#define EFI_ACPI_TABLE_GUID \
    { 0xeb9d2d30, 0x2d88, 0x11d3, {0x9a, 0x16, 0x0, 0x90, 0x27, 0x3f, 0xc1, 0x4d }}
#define EFI_ACPI_20_TABLE_GUID \
    { 0x8868e871, 0xe4f1, 0x11d3, {0xbc, 0x22, 0x0, 0x80, 0xc7, 0x3c, 0x88, 0x81 }}
```



```
#define EFI_ACPI_2_0_ROOT_SYSTEM_DESCRIPTION_POINTER_REVISION 0x02
#define EFI_SHELL_INTERFACE_GUID \
    {0x47c7b223, 0xc42a, 0x11d2, {0x8e,0x57,0x00,0xa0,0xc9,0x69,0x72,0x3b}}
#define SHELL_VARIABLE_GUID \
    {0x158def5a, 0xf656, 0x419c, {0xb0,0x27,0x7a,0x31,0x92,0xc0,0x79,0xd2}}
#undef DEBUG
typedef struct {
    CHAR8   Signature[8];
    UINT8   Checksum;
    UINT8   OemId[6];
    UINT8   Revision;
    UINT32  RsdtAddress;
    UINT32  Length;
    UINT64  XsdtAddress;
    UINT8   ExtendedChecksum;
    UINT8   Reserved[3];
} EFI_ACPI_2_0_ROOT_SYSTEM_DESCRIPTION_POINTER;
// There are many kinds of SDT. All SDT may be split into two parts.
// A common header and data which is different for each table.
typedef struct {
    CHAR8   Signature[4];
    UINT32  Length;
    UINT8   Revision;
    UINT8   Checksum;
    CHAR8   OemId[6];
    CHAR8   OemTableId[8];
    UINT32  OemRevision;
    CHAR8   CreatorId[4];
    UINT32  CreatorRevision;
} EFI_ACPI_SDT_HEADER;
// Microsoft Data Management table structure
typedef struct {
    EFI_ACPI_SDT_HEADER Header;
    UINT32  SlsVersion;
    UINT32  SlsReserved;
    UINT32  SlsDataType;
    UINT32  SlsDataReserved;
    UINT32  SlsDataLength;
    CHAR8   ProductKey[30];
} EFI_ACPI_MSDM;
typedef enum {
    ARG_NO_ATTRIB       = 0x0,
    ARG_IS_QUOTED       = 0x1,
    ARG_PARTIALLY_QUOTED  = 0x2,
    ARG_FIRST_HALF_QUOTED = 0x4,
    ARG_FIRST_CHAR_IS_ESC = 0x8
} EFI_SHELL_ARG_INFO_TYPES;
struct _EFI_SHELL_ARG_INFO {
    UINT32 Attributes;
} __attribute__((packed)) __attribute__((aligned (1)));
typedef struct _EFI_SHELL_ARG_INFO EFI_SHELL_ARG_INFO;
struct _EFI_SHELL_INTERFACE {
    EFI_HANDLE           ImageHandle;
    EFI_LOADED_IMAGE    *Info;
    CHAR16             **Argv;
    UINTN                Argc;
    CHAR16             **RedirArgv;
    UINTN                RedirArgc;
    EFI_FILE            *StdIn;
    EFI_FILE            *StdOut;
    EFI_FILE            *StdErr;
    EFI_SHELL_ARG_INFO  *ArgInfo;
    BOOLEAN              EchoOn;
} __attribute__((packed)) __attribute__((aligned (1)));
typedef struct _EFI_SHELL_INTERFACE EFI_SHELL_INTERFACE;
static EFI_STATUS
get_args(EFI_HANDLE image, UINTN *argc, CHAR16 ***argv)
{
```



```
    EFI_STATUS Status;
    EFI_SHELL_INTERFACE *shell;
    EFI_GUID gEfiShellInterfaceGuid = EFI_SHELL_INTERFACE_GUID;
    Status = uefi_call_wrapper(BS->OpenProtocol, 6,
                               image, &amp;gEfiShellInterfaceGuid,
                               (VOID **)&amp;shell, image, NULL,
                               EFI_OPEN_PROTOCOL_GET_PROTOCOL);
    if (EFI_ERROR(Status))
        return Status;
    *argc = shell->Argc;
    *argv = shell->Argv;
    Status = uefi_call_wrapper(BS->CloseProtocol, 4, image,
                               &amp;gEfiShellInterfaceGuid, image, NULL);
    return Status;
}
VOID
Ascii2UnicodeStr(CHAR8 *String, CHAR16 *UniString, UINT8 length)
{
    int len = length;
    while (*String != '&#92;&#48;' &amp;&amp; len > 0) {
        *(UniString++) = (CHAR16) *(String++);
        len--;
    }
    *UniString = '&#92;&#48;';
}
VOID
Guid2String(CHAR16 *Buffer, EFI_GUID *Guid)
{
    SPrint(Buffer, 0, L"%08x-%04x-%04x-%02x%02x-%02x%02x%02x%02x%02x%02x",
           Guid->Data1, Guid->Data2, Guid->Data3,
           Guid->Data4[0], Guid->Data4[1], Guid->Data4[2], Guid->Data4[3],
           Guid->Data4[4], Guid->Data4[5], Guid->Data4[6], Guid->Data4[7]);
}
static VOID
ParseMSDM(EFI_ACPI_MSDM *Msdm, int verbose)
{
    CHAR16 Buffer[100];
    Print(L"\n");
    if (verbose) {
        Ascii2UnicodeStr((CHAR8 *)(Msdm->Header.Signature), Buffer, 4);
        Print(L"Signature        : %s\n", Buffer);
        Print(L"Length           : %d\n", Msdm->Header.Length);
        Print(L"Revision         : %d\n", Msdm->Header.Revision);
        Print(L"Checksum         : %d\n", Msdm->Header.Checksum);
        Ascii2UnicodeStr((CHAR8 *)(Msdm->Header.OemId), Buffer, 6);
        Print(L"Oem ID           : %s\n", Buffer);
        Ascii2UnicodeStr((CHAR8 *)(Msdm->Header.OemTableId), Buffer, 8);
        Print(L"Oem Table ID     : %s\n", Buffer);
        Print(L"Oem Revision     : %d\n", Msdm->Header.OemRevision);
        Ascii2UnicodeStr((CHAR8 *)(Msdm->Header.CreatorId), Buffer, 4);
        Print(L"Creator ID       : %s\n", Buffer);
        Print(L"Creator Revision  : %d\n", Msdm->Header.CreatorRevision);
        Print(L"SLS Version      : %d\n", Msdm->SlsVersion);
        Print(L"SLS Reserved     : %d\n", Msdm->SlsReserved);
        Print(L"SLS Data Type    : %d\n", Msdm->SlsDataType);
        Print(L"SLS Data Reserved : %d\n", Msdm->SlsDataReserved);
        Print(L"SLS Data Length   : %d\n", Msdm->SlsDataLength);
    }
    Ascii2UnicodeStr((CHAR8 *)(Msdm->ProductKey), Buffer, 29);
    if (verbose) {
        Print(L"Product Key      : %s\n", Buffer);
    } else {
        Print(L"%s\n", Buffer);
    }
}
static int
ParseRSDP( EFI_ACPI_2_0_ROOT_SYSTEM_DESCRIPTION_POINTER *Rsdp, CHAR16* GuidStr, int verbos
e)
```



```
{
    EFI_ACPI_SDT_HEADER *Xsdt, *Entry;
    CHAR16 SigStr[20], OemStr[20];
    UINT32 EntryCount;
    UINT64 *EntryPtr;
    int Index;
#ifdef DEBUG
    Print(L"\n\nACPI GUID: %s\n", GuidStr);
#endif
    Ascii2UnicodeStr((CHAR8 *)(Rsdp->OemId), OemStr, 6);
#ifdef DEBUG
    Print(L"\nFound RSDP. Version: %d  OEM ID: %s\n", (int)(Rsdp->Revision), OemStr);
#endif
    if (Rsdp->Revision >= EFI_ACPI_2_0_ROOT_SYSTEM_DESCRIPTION_POINTER_REVISION) {
        Xsdt = (EFI_ACPI_SDT_HEADER *)(Rsdp->XsdtAddress);
    } else {
#ifdef DEBUG
        Print(L"ERROR: No ACPI XSDT table found.\n");
#endif
        return 1;
    }
    if (strncmpa("XSDT", (CHAR8 *)(VOID *)(Xsdt->Signature), 4)) {
#ifdef DEBUG
        Print(L"ERROR: Invalid ACPI XSDT table found.\n");
#endif
        return 1;
    }
    Ascii2UnicodeStr((CHAR8 *)(Xsdt->OemId), OemStr, 6);
    EntryCount = (Xsdt->Length - sizeof (EFI_ACPI_SDT_HEADER)) / sizeof(UINT64);
#ifdef DEBUG
    Print(L"Found XSDT. OEM ID: %s  Entry Count: %d\n\n", OemStr, EntryCount);
#endif
    EntryPtr = (UINT64 *)(Xsdt + 1);
    for (Index = 0; Index < EntryCount; Index++, EntryPtr++) {
        Entry = (EFI_ACPI_SDT_HEADER *)((UINTN)(*EntryPtr));
        Ascii2UnicodeStr((CHAR8 *)(Entry->Signature), SigStr, 4);
        if (!strncmpa("MSDM", (CHAR8 *)(Entry->Signature), 4)) {
            ParseMSDM((EFI_ACPI_MSDM *)((UINTN)(*EntryPtr)), verbose);
        }
    }
    return 0;
}
static void
Usage(void)
{
    Print(L"Usage: listmsdm [-v|--verbose]\n");
}
EFI_STATUS
efi_main (EFI_HANDLE image_handle, EFI_SYSTEM_TABLE *systab)
{
    EFI_CONFIGURATION_TABLE *ect = systab->ConfigurationTable;
    EFI_ACPI_2_0_ROOT_SYSTEM_DESCRIPTION_POINTER *Rsdp = NULL;
    EFI_GUID AcpiTableGuid = ACPI_TABLE_GUID;
    EFI_GUID Acpi2TableGuid = ACPI_20_TABLE_GUID;
    CHAR16 GuidStr[100];
    UINTN argc;
    CHAR16 **argv;
    EFI_STATUS Status = EFI_SUCCESS;
    int Index;
    int Verbose = 0;
    InitializeLib(image_handle, systab);
    Status = get_args(image_handle, &amp;argc, &amp;argv);
    if (EFI_ERROR(Status)) {
        Print(L"ERROR: Parsing command line arguments: %d\n", Status);
        return Status;
    }
    if (argc == 2) {
        if (!StrCmp(argv[1], L"--verbose") ||
```



```
            !StrCmp(argv[1], L"-v")) {
            Verbose = 1;
        }
        if (!StrCmp(argv[1], L"--help") ||
            !StrCmp(argv[1], L"-h") ||
            !StrCmp(argv[1], L"-?")) {
            Usage();
            return Status;
        }
    }
    // locate RSDP (Root System Description Pointer)
    for (Index = 0; Index < systab->NumberOfTableEntries; Index++) {
        if ((CompareGuid (&amp;(systab->ConfigurationTable[Index].VendorGuid), &amp;AcpiTa
bleGuid)) ||
            (CompareGuid (&amp;(systab->ConfigurationTable[Index].VendorGuid), &amp;Acpi2T
ableGuid))) {
            if (!strncmpa("RSD PTR ", (CHAR8 *)(ect->VendorTable), 8)) {
                Guid2String(GuidStr, &amp;(systab->ConfigurationTable[Index].VendorGuid));
                Rsdp = (EFI_ACPI_2_0_ROOT_SYSTEM_DESCRIPTION_POINTER *)ect->VendorTable;
                ParseRSDP(Rsdp, GuidStr, Verbose);
            }
        }
        ect++;
    }
    if (Rsdp == NULL) {
        if (Verbose) {
            Print(L"ERROR: Could not find an ACPI RSDP table.\n");
        }
        return EFI_NOT_FOUND;
    }
    return Status;
}
```

As with other UEFI utilities that I have published on this blog, it uses the GNU EFI library. I assume you are familiar with ACPI tables and UEFI APIs if you are reading this post and thus make no attempt to explain the source code.

Here is a suitable *Makefile* to build *listmsdm.efi*:

```
SRCDIR     = .
PREFIX     := /usr
HOSTARCH   := $(shell uname -m | sed s,i[3456789]86,ia32,)
ARCH       := $(shell uname -m | sed s,i[3456789]86,ia32,)
INCDIR     = -I.
CPPFLAGS   = -DCONFIG_$(ARCH)
CFLAGS     = $(ARCH3264) -g -O0 -fpic -Wall -fshort-wchar -fno-strict-aliasing -fno-merge-
constants --std=gnu99 -D_GNU_SOURCE
ASFLAGS    = $(ARCH3264)
LDFLAGS    = -nostdlib
INSTALL    = install

CC         = gcc
AS         = as
LD         = ld.bfd
AR         = ar
RANLIB     = ranlib
OBJCOPY    = objcopy

ifeq ($(ARCH), ia32)
  LIBDIR := $(PREFIX)/lib
  ifeq ($(HOSTARCH), x86_64)
    ARCH3264 := -m32
  endif
endif
```



```
ifeq ($(ARCH), x86_64)
  CFLAGS += -mno-red-zone
  LIBDIR := $(PREFIX)/lib64
  ifeq ($(HOSTARCH), ia32)
    ARCH3264 := -m64
  endif
endif

FORMAT=efi-app-$(HOSTARCH)
LDFLAGS = -nostdlib -T $(LIBDIR)/gnuefi/elf_$(HOSTARCH)_efi.lds -shared -Bsymbolic $(LIBDI
R)/gnuefi/crt0-efi-$(HOSTARCH).o -L$(LIBDIR)
LIBS=-lefi -lgnuefi $(shell $(CC) -print-libgcc-file-name)
CCLDFLAGS =
CFLAGS = -I/usr/include/efi/ -I/usr/include/efi/$(HOSTARCH)/ -I/usr/include/efi/protocol -
fpic -fshort-wchar -fno-reorder-functions -fno-strict-aliasing -fno-merge-constants -mno-r
ed-zone -Wimplicit-function-declaration


TARGETS = listmsdm.efi

all : $(TARGETS)

clean :
    @rm -rf *.o *.a *.so $(TARGETS)

.PHONY: all clean install


%.efi : %.so
    $(OBJCOPY) -j .text -j .sdata -j .data -j .dynamic -j .dynsym -j .rel \
        -j .rela -j .reloc --target=$(FORMAT) $*.so $@

%.so: %.o
    $(LD) $(LDFLAGS) -o $@ $^ $(LIBS)

%.o: %.c
    $(CC) $(INCDIR) $(CFLAGS) $(CPPFLAGS) -D__UEFI__ -c $< -o $@

%.S: %.c
    $(CC) $(INCDIR) $(CFLAGS) $(CPPFLAGS) -D__UEFI__ -S $< -o $@

%.E: %.c
    $(CC) $(INCDIR) $(CFLAGS) $(CPPFLAGS) -D__UEFI__ -E $< -o $@
```

Note that I have not tested a 32-bit build.

There are a number of utilities available that can retrieve the product key from within Microsoft Windows but as far as I know nobody to date has retrieved this information directly via the UEFI shell using a custom EFI utility. This utility, by the way, has been included in version 0.3 of my UEFI Rescue DVD (URD).

