Examine ESRT entries from UEFI Shell

Finnbarr P. Murphy
(fpm@fpmurphy.com)

This post details a simple UEFI shell utility for listing the contents of an EFI System Resource Table (ESRT). Essentially ESRT a catalog of firmware which can be updated with the UEFI *UpdateCapsule* mechanism described in section 7.5 of the UEFI Specification.

The ESRT provides a mechanism for identifying integrated device and system firmware resources for the purposes
of targeting firmware updates to those resources. Each entry in the ESRT describes a device or system firmware resource that can be targeted by a firmware update package. UEFI firmware must allocate and populate an ESRT system resource entry for itself (system firmware). This entry is used to target a system firmware update.

In addition, each firmware resource (typically a UEFI device driver) that can be updated by a firmware update package must be described by exactly one entry in the ESRT to enable firmware updates to be deployed and installed. If an implementation performs system and device firmware updates as a single, monolithic operation, the system firmware entry must be used to target the update. In all other cases, device firmware updates are targeted by an ESRT entry describing device firmware.

Here is a table (from a Microsoft document) detailing the structure of ESRT header and entries:

Field	Byte Length	Byte Offset	Description
Firmware Resource Count	4	0	The number of firmware resources, must not be zero.
Firmware Resource Maximum	4	4	The maximum number of resource array entries that can be added without reallocating the table, must not be zero.
Firmware Resource Version	8	8	The firmware resource entry version.
Firmware Resource Entry Array			Firmware Resource Entry 0
Firmware Class	16	16	This GUID defines the class of systems for which an update gets applied.
Firmware Type	4	32	Identifies the type of firmware resource.
Firmware Version	4	36	The current firmware version, where a larger number represents a newer release. The value is not defined but should incorporate version major and minor numbers. Recommended format is first word is major and second word is minor version numbers.
Lowest Supported Firmware Version	4	40	The lowest firmware version that can be rolled back to, i.e. the last firmware version that contained a security fix.
Capsule Flags	4	44	The OS will set the upper capsule flags defined by the UEFI spec, but the low 16-bits are defined per capsule.
Last Attempt Version	4	48	Version of the last firmware update attempt.
Last Attempt Status	4	52	Status of the last firmware update attempt.
...			Firmware Resource Entry 1

Possible causes for firmware update failure include, but are not limited to:

- Insufficient resources
- Power loss
- Hardware failure
- Firmware incompatible with OS drivers
- Firmware incompatible with OS components

I assume you are familiar with EDK2, UDK2015 and GNU EFI if you are reading this blog post and know how to install the appropriate toolchains to build EFI images. Therefore, I am not going to describe how to install and maintain these toolchains but simply present the source code and corresponding build file for the utility.

Here is the source code and .INF file for building the utility using UDK2015:

```
//
//  Copyright (c) 2015  Finnbarr P. Murphy.   All rights reserved.
//
//  Display ESRT entries
```



```
//
//  License: BSD License
//
#include <Uefi.h>
#include <Library/UefiLib.h>
#include <Library/ShellCEntryLib.h>
#include <Library/ShellLib.h>
#include <Library/BaseMemoryLib.h>
#include <Library/UefiBootServicesTableLib.h>
#include <Protocol/EfiShell.h>
#include <Protocol/LoadedImage.h>
#define ESRT_GUID { 0xb122a263, 0x3661, 0x4f68, { 0x99, 0x29, 0x78, 0xf8, 0xb0, 0xd6, 0x21,
 0x80 }}
EFI_GUID EsrtGuid = ESRT_GUID;
typedef struct esrt {
    UINT32 fw_resource_count;
    UINT32 fw_resource_count_max;
    UINT64 fw_resource_version;
} __attribute__((__packed__)) esrt_t;
typedef struct esre1 {
    EFI_GUID fw_class;
    UINT32   fw_type;
    UINT32   fw_version;
    UINT32   lowest_supported_fw_version;
    UINT32   capsule_flags;
    UINT32   last_attempt_version;
    UINT32   last_attempt_status;
} __attribute__((__packed__)) esre1_t;
#define CAPSULE_FLAGS_PERSIST_ACROSS_RESET  0x00010000
#define CAPSULE_FLAGS_POPULATE_SYSTEM_TABLE 0x00020000
#define CAPSULE_FLAGS_INITIATE_RESET        0x00040000
void
dump_esrt(VOID *data)
{
    esrt_t *esrt = data;
    esre1_t *esre1 = (esre1_t *)((UINT8 *)data + sizeof (*esrt));
    int i, ob;
    Print(L"Dumping ESRT found at 0x%x\n\n", data);
    Print(L"Firmware Resource Count: %d\n", esrt->fw_resource_count);
    Print(L"Firmware Resource Max Count: %d\n", esrt->fw_resource_count_max);
    Print(L"Firmware Resource Version: %ld\n\n", esrt->fw_resource_version);
    if (esrt->fw_resource_version != 1) {
        Print(L"ERROR: Unknown ESRT version: %d\n", esrt->fw_resource_version);
        return;
    }
    for (i = 0; i < esrt->fw_resource_count; i++) {
        ob = 0;
        Print(L"ENTRY NUMBER: %d\n", i);
        Print(L"Firmware Class GUID: %g\n", &amp;esre1->fw_class);
        Print(L"Firmware Type: %d ", esre1->fw_type);
        switch (esre1->fw_type) {
            case 0:  Print(L"(Unknown)\n");
                     break;
            case 1:  Print(L"(System)\n");
                     break;
            case 2:  Print(L"(Device)\n");
                     break;
            case 3:  Print(L"(UEFI Driver)\n");
                     break;
            default: Print(L"\n");
        }
        Print(L"Firmware Version: 0x%08x\n", esre1->fw_version);
        Print(L"Lowest Supported Firmware Version: 0x%08x\n", esre1->lowest_supported_fw_v
ersion);
        Print(L"Capsule Flags: 0x%08x", esre1->capsule_flags);
        if ((esre1->capsule_flags &amp; (CAPSULE_FLAGS_PERSIST_ACROSS_RESET)) == CAPSULE_F
LAGS_PERSIST_ACROSS_RESET) {
            if (!ob) {
```



```
                ob = 1;
                Print(L"(");
            }
            Print(L"PERSIST");
        }
        if ((esre1->capsule_flags &amp;amp; (CAPSULE_FLAGS_POPULATE_SYSTEM_TABLE)) == CAPSULE_
FLAGS_POPULATE_SYSTEM_TABLE) {
            if (!ob) {
                ob = 1;
                Print(L"(");
            } else
                Print(L", ");
            Print(L"POPULATE");
        }
        if ((esre1->capsule_flags &amp;amp; (CAPSULE_FLAGS_INITIATE_RESET)) == CAPSULE_FLAGS_I
NITIATE_RESET) {
            if (!ob) {
                ob = 1;
                Print(L"(");
            } else
                Print(L", ");
            Print(L"RESET");
        }
        if (ob)
            Print(L")");
        Print(L"\n");
        Print(L"Last Attempt Version: 0x%08x\n", esre1->last_attempt_version);
        Print(L"Last Attempt Status: %d ", esre1->last_attempt_status);
        switch(esre1->last_attempt_status) {
            case 0:  Print(L"(Success)\n");
                     break;
            case 1:  Print(L"(Unsuccessful)\n");
                     break;
            case 2:  Print(L"(Insufficient Resources)\n");
                     break;
            case 3:  Print(L"(Incorrect version)\n");
                     break;
            case 4:  Print(L"(Invalid Format)\n");
                     break;
            case 5:  Print(L"(AC Power Issue)\n");
                     break;
            case 6:  Print(L"(Battery Power Issue)\n");
                     break;
            default: Print(L"\n");
        }
        Print(L"\n");
        esre1++;
    }
}
INTN
EFIAPI
ShellAppMain(UINTN Argc, CHAR16 **Argv)
{
    EFI_CONFIGURATION_TABLE *ect = gST->ConfigurationTable;
    for (int i = 0; i < gST->NumberOfTableEntries; i++) {
        if (!CompareMem(&amp;ect->VendorGuid, &amp;EsrtGuid, sizeof(EsrtGuid))) {
            dump_esrt(ect->VendorTable);
            return EFI_SUCCESS;
        }
        ect++;
        continue;
    }
    Print(L"No ESRT found\n");
    return EFI_SUCCESS;
}
```



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

[Sources]
  ShowESRT.c

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


[LibraryClasses]
  ShellCEntryLib
  ShellLib
  BaseLib
  BaseMemoryLib
  UefiLib

[Protocols]

[BuildOptions]

[Pcd]
```

Here is the source code and Makefile for building the utility using *GNU EFI*:

```
//
//  Copyright (c) 2015  Finnbarr P. Murphy.   All rights reserved.
//
//  Display ESRT entries
//
//  License: BSD License
//
#include <efi.h>
#include <efilib.h>
#define ESRT_GUID { 0xb122a263, 0x3661, 0x4f68, { 0x99, 0x29, 0x78, 0xf8, 0xb0, 0xd6, 0x21,
 0x80 }}
EFI_GUID EsrtGuid = ESRT_GUID;
typedef struct esrt {
    UINT32 fw_resource_count;
    UINT32 fw_resource_count_max;
    UINT64 fw_resource_version;
} __attribute__((__packed__)) esrt_t;
typedef struct esre1 {
    EFI_GUID fw_class;
    UINT32   fw_type;
    UINT32   fw_version;
    UINT32   lowest_supported_fw_version;
    UINT32   capsule_flags;
    UINT32   last_attempt_version;
    UINT32   last_attempt_status;
} __attribute__((__packed__)) esre1_t;
#define CAPSULE_FLAGS_PERSIST_ACROSS_RESET  0x00010000
#define CAPSULE_FLAGS_POPULATE_SYSTEM_TABLE 0x00020000
#define CAPSULE_FLAGS_INITIATE_RESET        0x00040000
void
dump_esrt(VOID *data)
```

```
{
    esrt_t *esrt = data;
    esre1_t *esre1 = (esre1_t *)((UINT8 *)data + sizeof (*esrt));
    int i, ob;
    Print(L"Dumping ESRT found at 0x%x\n\n", data);
    Print(L"Firmware Resource Count: %d\n", esrt->fw_resource_count);
    Print(L"Firmware Resource Max Count: %d\n", esrt->fw_resource_count_max);
    Print(L"Firmware Resource Version: %ld\n\n", esrt->fw_resource_version);
    if (esrt->fw_resource_version != 1) {
        Print(L"ERROR: Unknown ESRT version: %d\n", esrt->fw_resource_version);
        return;
    }
    for (i = 0; i < esrt->fw_resource_count; i++) {
        ob = 0;
        Print(L"ENTRY NUMBER: %d\n", i);
        Print(L"Firmware Class GUID: %g\n", &amp;esre1->fw_class);
        Print(L"Firmware Type: %d ", esre1->fw_type);
        switch (esre1->fw_type) {
            case 0:  Print(L"(Unknown)\n");
                     break;
            case 1:  Print(L"(System)\n");
                     break;
            case 2:  Print(L"(Device)\n");
                     break;
            case 3:  Print(L"(UEFI Driver)\n");
                     break;
            default: Print(L"\n");
        }
        Print(L"Firmware Version: 0x%08x\n", esre1->fw_version);
        Print(L"Lowest Supported Firmware Version: 0x%08x\n", esre1->lowest_supported_fw_v
ersion);
        Print(L"Capsule Flags: 0x%08x", esre1->capsule_flags);
        if ((esre1->capsule_flags &amp; (CAPSULE_FLAGS_PERSIST_ACROSS_RESET)) == CAPSULE_F
LAGS_PERSIST_ACROSS_RESET) {
            if (!ob) {
                ob = 1;
                Print(L"(");
            }
            Print(L"PERSIST");
        }
        if ((esre1->capsule_flags &amp; (CAPSULE_FLAGS_POPULATE_SYSTEM_TABLE)) == CAPSULE_
FLAGS_POPULATE_SYSTEM_TABLE) {
            if (!ob) {
                ob = 1;
                Print(L"(");
            } else
                Print(L", ");
            Print(L"POPULATE");
        }
        if ((esre1->capsule_flags &amp; (CAPSULE_FLAGS_INITIATE_RESET)) == CAPSULE_FLAGS_I
NITIATE_RESET) {
            if (!ob) {
                ob = 1;
                Print(L"(");
            } else
                Print(L", ");
            Print(L"RESET");
        }
        if (ob)
            Print(L")");
        Print(L"\n");
        Print(L"Last Attempt Version: 0x%08x\n", esre1->last_attempt_version);
        Print(L"Last Attempt Status: %d ", esre1->last_attempt_status);
        switch(esre1->last_attempt_status) {
            case 0:  Print(L"(Success)\n");
                     break;
            case 1:  Print(L"(Unsuccessful)\n");
                     break;
```



```
            case 2:  Print(L"(Insufficient Resources)\n");
                     break;
            case 3:  Print(L"(Incorrect version)\n");
                     break;
            case 4:  Print(L"(Invalid Format)\n");
                     break;
            case 5:  Print(L"(AC Power Issue)\n");
                     break;
            case 6:  Print(L"(Battery Power Issue)\n");
                     break;
            default: Print(L"\n");
        }
        Print(L"\n");
        esre1++;
    }
}
EFI_STATUS
efi_main (EFI_HANDLE image_handle, EFI_SYSTEM_TABLE *systab)
{
    EFI_CONFIGURATION_TABLE *ect = systab->ConfigurationTable;
    InitializeLib(image_handle, systab);
    for (int i = 0; i < systab->NumberOfTableEntries; i++) {
        if (!CompareMem(&amp;ect->VendorGuid, &amp;EsrtGuid, sizeof(EsrtGuid))) {
            dump_esrt(ect->VendorTable);
            return EFI_SUCCESS;
        }
        ect++;
        continue;
    }
    Print(L"No ESRT found\n");
    return EFI_SUCCESS;
}
```

```
TARGET     = showesrt.efi
SRCS       = showesrt.c

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)
ASFLAGS    = $(ARCH3264)
LDFLAGS    = -nostdlib
INSTALL    = install

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

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

OBJS = $(SRCS:.c=.o)

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


all : $(TARGET)

.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 $@

$(TARGET:.efi=.so): $(OBJS)
    $(LD) $(LDFLAGS) -o $@ $^ $(LIBS)

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

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

I tested both builds using 64-bit Fedora 23. Neither build has been tested on 32-bit architectures nor has been fully tested for flawed login or bugs.

Here is the output I got on my Lenovo T450 laptop:

```
Dumping ESRT found at 0xBAAEA000

Firmware Resource Count: 2
Firmware Resource Max Count: 2
Firmware Resource Version: 1

ENTRY NUMBER: 0
Firmware Class GUID: DE431F21-4606-4787-B426-25A77C5B9B46
Firmware Type: 1 (System)
Firmware Version: 0x00010012
Lowest Supported Firmware Version: 0x00010012
Capsule Flags: 0x00000000
Last Attempt Version: 0x01497000
Last Attempt Status: 0 (Success)

ENTRY NUMBER: 1
Firmware Class GUID: FFEC4692-FF4F-4D19-A311-453F50256192
Firmware Type: 2 (Device)
Firmware Version: 0xA01E0430
Lowest Supported Firmware Version: 0xA01E0430
Capsule Flags: 0x00008010
Last Attempt Version: 0x00000000
Last Attempt Status: 0 (Success)
```

If you want further information, Microsoft has some good documentation on the ESRT and firmware updating in Windows 8 and later. Do an Internet search for "Windows UEFI Firmware Update Platform."

Happy 2016!

