

EFI/UEFI Programming with UASM 2.50+

UASM v2.50 includes several updates to support interface-based calling against non-COM or OO structures. Making use of these features a large portion of the UEFI core headers have been ported to work with UASM.

This guide serves as a quick introduction on how to get started working with UEFI from Assembly Language and specifically UASM.

Setting up a Virtual Machine

For testing and development purposes I recommend installing the latest version of Oracle Virtual Box (6.0). This will give you a Virtual Machine that you can boot to test your UEFI application, driver or OS. <https://www.virtualbox.org/wiki/Downloads>

To keep the configuration simple, we will use a USB flash drive rather than configure a UEFI bootable VDI for Virtual Box. This has the benefit of not only being much easier to setup, but you can test the USB stick on real machines too.

Get the UEFI EDK2 SDK and Tools

Once you have Virtual Box up and running with a new VM, you'll want to download a copy of the EDK (UEFI development kit). I would recommend using a stable release from:

<https://github.com/tianocore/tianocore.github.io/wiki/EDK-II>

All the latest UEFI specifications and tools are available from:

<https://uefi.org/specsandtesttools>

You will also want the FWIMAGE utility from the EFI Toolkit (It has been discontinued but the application is still valid and available).

<https://github.com/tianocore/tianocore.github.io/wiki/EFI-Toolkit>

Once extracted the utility can be found at:

`\\EFI_Toolkit_2.0.0.1\\EFI_Toolkit_2.0\\build\\tools\\bin`

It is used to modify a normal PE32+ DLL/EXE to have the correct EFI subsystem type:

`D:\\UEFI\\EFI_Toolkit_2.0.0.1\\EFI_Toolkit_2.0\\build\\tools\\bin\\fwimage app uefi.dll app.EFI`

and will form part of your build script later.

Prepare the USB Flash Driver

Format the USB Flash drive as FAT32. Create a folder structure `\EFI\BOOT\` on the driver. This is the default location for the firmware to look for the initial EFI application to load. The application / depending on your architecture will be named something like `BOOTX64.EFI`. For our example we will assume a 64bit UEFI PC and use the above name.

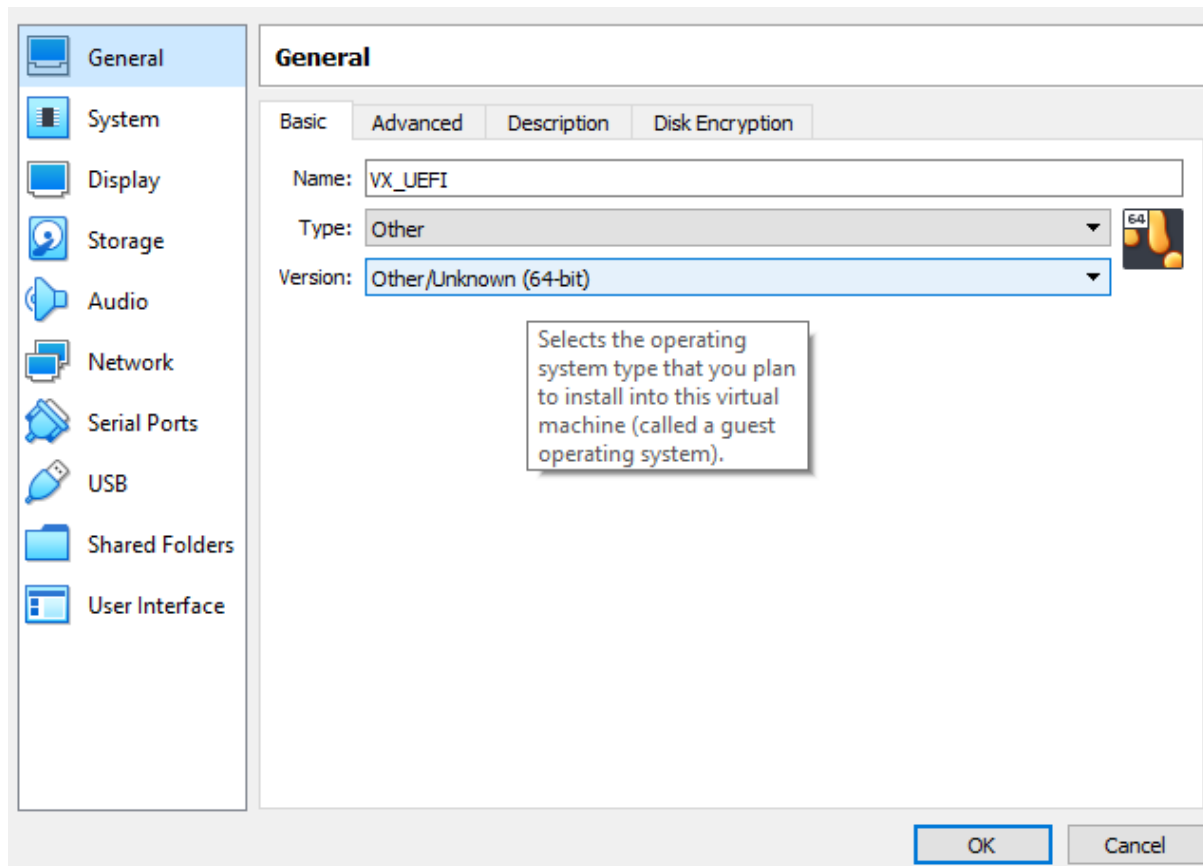
The UEFI EDK includes an EFI Shell application, we will make this the default application to load when the USB stick is booted.

Copy the file from your UEFI installation `\UEFI\ShellBinPkg\ShellBinPkg\UefiShell\X64\Shell.efi`

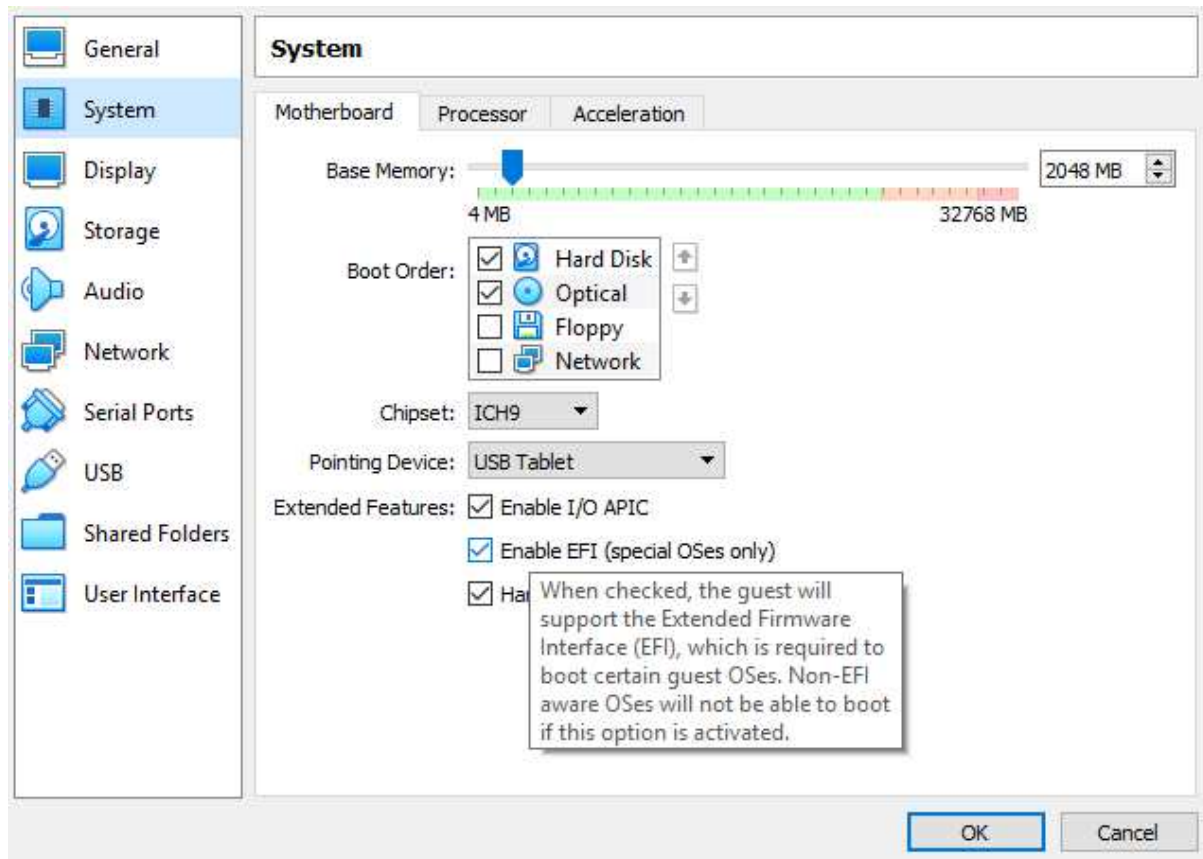
To the `/EFI/BOOT/` folder you created earlier on the USB stick. Rename the file to **BOOTX64.EFI**.

Configure the VM

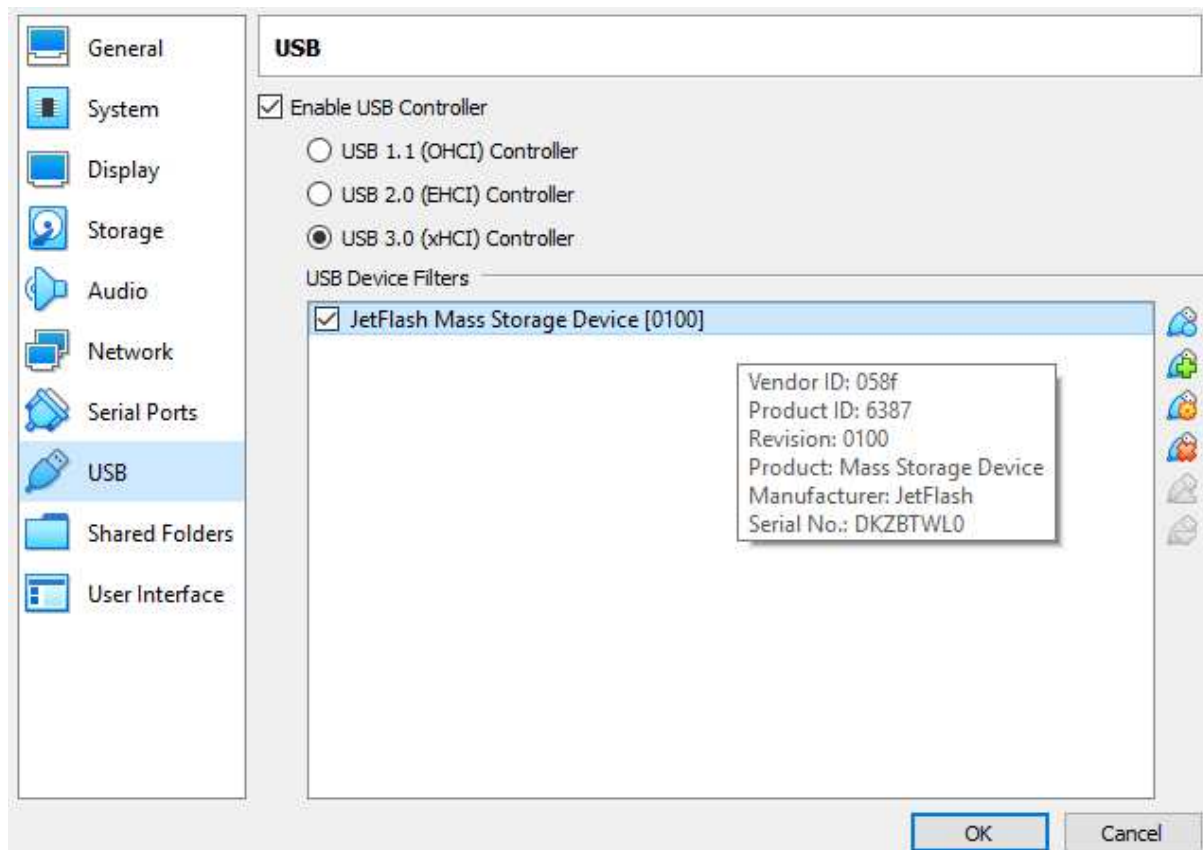
Ensure that the VM is configured as 64bit other:

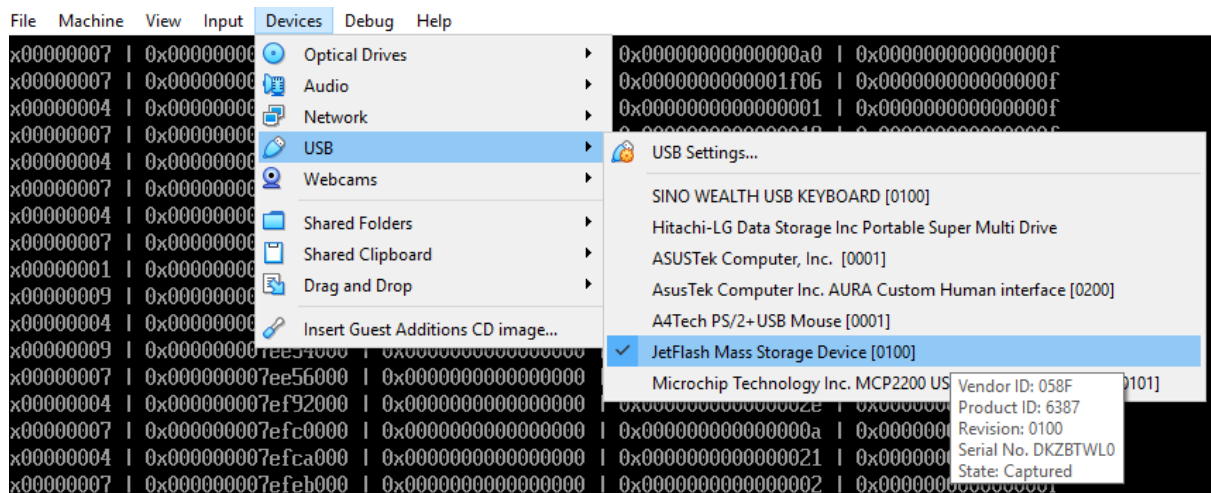


Ensure the VM is configured to use EFI:



Setup USB pass-through to share the real USB stick with the VM:





The UASM UEFI Includes

The entire UEFI core is made available via a single *efi.inc* include file.

This file includes all the structures, equates and types defined in the UEFI core includes as well as the majority of Protocols and GUIDs. For details on the protocols and features please refer to the UEFI Specification which includes detailed coverage of how the protocols are located, used as well as full method and data type descriptions.

In addition a *efiUtil.inc* file is provided with several helper functions for some common UEFI operations. These functions are available as regular procedure calls or through a similar pointer based interface as the core UEFI functions:

```

RAWINTERFACE iEFIUtil

    STDFUNC PrintMemoryDescriptor, <voidarg>, pConsole:PCONOUT, pDescriptor:PTR
EFI_MEMORY_DESCRIPTOR

    STDFUNC PrintGraphicsModeInfo, <voidarg>, pConsole:PCONOUT, ModeNumber:DWORD, pGfxMode:PTR
EFIGraphicsMode, showHeader:BOOLEAN

    STDFUNC CompareGUID, <voidarg>, guidA:PTR, guidB:PTR

ENDRAWINTERFACE

PEFIUtil TYPEDEF PTR iEFIUtil

```

Evolution of Calls

In a primitive assembler, assuming the UEFI structures had been provided, and as many online examples might show to make an ABI compliant FASTCALL from assembly language would require code similar to:

```
=====
; The normal ASM way to make a 64bit FASTCALL.
=====
    sub rsp,20h
    lea rdx,HelloMsg
    mov rcx,SystemTablePtr
    mov rcx,[rcx + EFI_SYSTEM_TABLE_CONOUT]
    call qword ptr [rcx + EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL_OUTPUTSTRING]
    add rsp,20h
```

But we can do much better than that!

By using INVOKE syntax and the fact that UASM support built in UNICODE string literals:

```
=====
; The smarter UASM way...
=====
    mov rcx,SystemTablePtr
    mov rax,[rcx].EFI_SYSTEM_TABLE.ConIn
    mov pConsoleIn,rax
    mov rax,[rcx].EFI_SYSTEM_TABLE.ConOut
    mov pConsole,rax
    invoke [rax].ConOut.OutputString, pConsole, L"Hello Smarter UEFI World!\r\n"

; Or if you have a list of calls to make against the same protocol/interface
ASSUME rcx:PTR ConOut
    mov rcx,pConsole
    invoke [rcx].OutputString, pConsole, ADDR HelloMsg
ASSUME rcx:NOTHING
```

Now making use of UASM 2.50+ enhanced interface calling and the types that have been provided by the include file:

```
=====
; The even smarter ways...
=====
    pConsole->OutputString(pConsole, L"Testing\r\n")
; or
    mov rax,pConsole
    [rax].ConOut->OutputString(pConsole, L"Testing2\r\n")

    pConsole->ClearScreen(pConsole)
```

Hello World UEFI!

Using some of the available features from UEFI we can now setup a minimal hello world UEFI example source:

```
.x64p
OPTION WIN64:15
OPTION STACKBASE:RSP
OPTION LITERALS:ON
OPTION ARCH:AVX
OPTION CASEMAP:NONE

include efi.inc

.data

Handle          EFI_HANDLE 0
SystemTablePtr  dq 0
HelloMsg        dw 'Hello UEFI World!',13,10,0
pConsole        PCONOUT 0
pConsoleIn      PCONIN 0
pBootServices   P_BOOT_SERVICES 0
pRuntimeServices P_RUNTIME_SERVICES 0

mapSize        UINTN 512*SIZEOF(EFI_MEMORY_DESCRIPTOR)
descriptors     EFI_MEMORY_DESCRIPTOR 512 DUP (<?>)
mapKey          UINTN 0
descSize        UINTN 0
descVer         UINT32 0

include efiUtil.inc

.code

Main PROC FRAME imageHandle:EFI_HANDLE, SystemTable:PTR_EFI_SYSTEM_TABLE

    mov Handle,rcx
    mov SystemTablePtr,rdx

;=====
; The normal ASM way to make a 64bit FASTCALL.
;=====
    sub rsp,20h
    lea rdx,HelloMsg
    mov rcx,SystemTablePtr
    mov rcx,[rcx + EFI_SYSTEM_TABLE_CONOUT]
    call qword ptr [rcx + EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL_OUTPUTSTRING]
    add rsp,20h

;=====
; The smarter UASM way...
;=====
    mov rcx,SystemTablePtr
    mov rax,[rcx].EFI_SYSTEM_TABLE.ConIn
    mov pConsoleIn,rax
    mov rax,[rcx].EFI_SYSTEM_TABLE.ConOut
    mov pConsole,rax
    invoke [rax].ConOut.OutputString, pConsole, L"Hello Smarter UEFI World!\r\n"
```

```

; Or if you have a list of calls to make against the same protocol/interface
ASSUME rcx:PTR ConOut
mov rcx,pConsole
invoke [rcx].OutputString, pConsole, ADDR HelloMsg
ASSUME rcx:NOTHING

;=====
; The even smarter ways...
;=====
    pConsole->OutputString(pConsole, L"Testing\r\n")
    ; or
    mov rax,pConsole
    [rax].ConOut->OutputString(pConsole, L"Testing2\r\n")

    pConsole->ClearScreen(pConsole)

;=====
; Store pointer to the BOOT SERVICES and RUNTIME SERVICES Interfaces
;=====
    mov rax,SystemTablePtr
    mov rsi,[rax].EFI_SYSTEM_TABLE.RuntimeServices
    mov pRuntimeServices,rsi
    mov rsi,[rax].EFI_SYSTEM_TABLE.BootServices
    mov pBootServices,rsi

;=====
; Get Memory Map
;=====
    [rsi].BOOT_SERVICES->GetMemoryMap(&mapSize, &descriptors, &mapKey, &descSize,
&descVer)
    .if(rax != EFI_SUCCESS)
        pConsole->OutputString(pConsole, L"Failed to get memory map\r\n")
        pBootServices->Exit(Handle, EFI_ERROR, 36, L"Memory Map Error\r\n")
    .else
        lea rsi,descriptors
        mov rax,mapSize
        xor rdx,rdx
        idiv descSize
        mov r11,rax
        .for(r10=0 : r10 < r11 : r10++)
            pEFIUtil->PrintMemoryDescriptor(pConsole, rsi)
            add rsi,descSize
        .endfor
    .endif

    mov eax,EFI_SUCCESS
    pBootServices->Exit(Handle, EFI_SUCCESS, 10, L"Complete\r\n")

    ret
Main ENDP

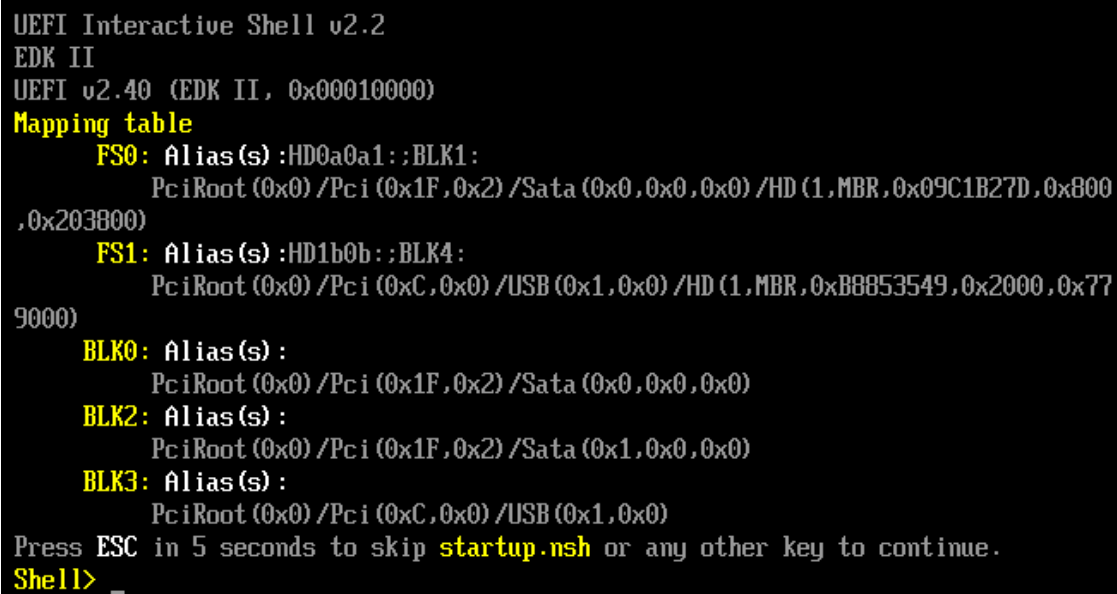
END Main

```

An Example Build Script

```
uasm64 -c -win64 -Zp8 uefi.asm
link /dll /IGNORE:4086 uefi.obj
fwimage app uefi.dll app.EFI
copy app.EFI f:\EFI\BOOT\app.EFI
```

Running the Example



```
UEFI Interactive Shell v2.2
EDK II
UEFI v2.40 (EDK II, 0x00010000)
Mapping table
  FS0: Alias(s) :HD0a0a1:;BLK1:
      PciRoot (0x0) /Pci (0x1F,0x2) /Sata (0x0,0x0,0x0) /HD (1,MBR,0x09C1B27D,0x800
,0x203800)
  FS1: Alias(s) :HD1b0b:;BLK4:
      PciRoot (0x0) /Pci (0xC,0x0) /USB (0x1,0x0) /HD (1,MBR,0xB8853549,0x2000,0x77
9000)
  BLK0: Alias(s) :
      PciRoot (0x0) /Pci (0x1F,0x2) /Sata (0x0,0x0,0x0)
  BLK2: Alias(s) :
      PciRoot (0x0) /Pci (0x1F,0x2) /Sata (0x1,0x0,0x0)
  BLK3: Alias(s) :
      PciRoot (0x0) /Pci (0xC,0x0) /USB (0x1,0x0)
Press ESC in 5 seconds to skip startup.nsh or any other key to continue.
Shell> _
```

Once your VM boots you should be presented with the UEFI Shell. From the listed of available file systems navigate to the USB stick, in our case this is **fs1**. The shell operates in a very similar fashion to DOS or Windows shell. Execute the below steps replacing the volume as applicable:

```
fs1:
cd efi
cd boot
app.efi { > out.txt }
```

Which should give you and output like:


```
0x00000007 | 0x0000000000000000 | 0x0000000000000000 | 0x00000000000000a0 | 0x000000000000000f
0x00000007 | 0x0000000000100000 | 0x0000000000000000 | 0x0000000000001f06 | 0x000000000000000f
0x00000004 | 0x0000000002006000 | 0x0000000000000000 | 0x0000000000000001 | 0x000000000000000f
0x00000007 | 0x0000000002007000 | 0x0000000000000000 | 0x0000000000000019 | 0x000000000000000f
0x00000004 | 0x0000000002020000 | 0x0000000000000000 | 0x000000000000008e0 | 0x000000000000000f
0x00000007 | 0x0000000002900000 | 0x0000000000000000 | 0x00000000000079700 | 0x000000000000000f
0x00000004 | 0x0000000007c00000 | 0x0000000000000000 | 0x0000000000000020 | 0x000000000000000f
0x00000007 | 0x0000000007c02000 | 0x0000000000000000 | 0x000000000000028c2 | 0x000000000000000f
0x00000001 | 0x0000000007e8e2000 | 0x0000000000000000 | 0x00000000000000f0 | 0x000000000000000f
0x00000009 | 0x0000000007e9d2000 | 0x0000000000000000 | 0x0000000000000001 | 0x000000000000000f
0x00000004 | 0x0000000007e9d3000 | 0x0000000000000000 | 0x00000000000000481 | 0x000000000000000f
0x00000009 | 0x0000000007ee54000 | 0x0000000000000000 | 0x0000000000000002 | 0x000000000000000f
0x00000007 | 0x0000000007ee56000 | 0x0000000000000000 | 0x0000000000000013c | 0x000000000000000f
0x00000004 | 0x0000000007ef92000 | 0x0000000000000000 | 0x000000000000002e | 0x000000000000000f
0x00000007 | 0x0000000007efc0000 | 0x0000000000000000 | 0x000000000000000a | 0x000000000000000f
0x00000004 | 0x0000000007efca000 | 0x0000000000000000 | 0x0000000000000021 | 0x000000000000000f
0x00000007 | 0x0000000007efeb000 | 0x0000000000000000 | 0x0000000000000002 | 0x000000000000000f
0x00000004 | 0x0000000007efed000 | 0x0000000000000000 | 0x0000000000000001 | 0x000000000000000f
0x00000007 | 0x0000000007efee000 | 0x0000000000000000 | 0x000000000000000a | 0x000000000000000f
0x00000004 | 0x0000000007eff8000 | 0x0000000000000000 | 0x000000000000000a | 0x000000000000000f
0x00000007 | 0x0000000007f002000 | 0x0000000000000000 | 0x0000000000000001 | 0x000000000000000f
0x00000004 | 0x0000000007f003000 | 0x0000000000000000 | 0x00000000000000121 | 0x000000000000000f
0x00000007 | 0x0000000007f124000 | 0x0000000000000000 | 0x0000000000000001 | 0x000000000000000f
0x00000004 | 0x0000000007f125000 | 0x0000000000000000 | 0x0000000000000060 | 0x000000000000000f
0x00000007 | 0x0000000007f185000 | 0x0000000000000000 | 0x0000000000000001 | 0x000000000000000f
0x00000004 | 0x0000000007f186000 | 0x0000000000000000 | 0x00000000000000bd0 | 0x000000000000000f
0x00000007 | 0x0000000007fd56000 | 0x0000000000000000 | 0x0000000000000012 | 0x000000000000000f
0x00000003 | 0x0000000007fd68000 | 0x0000000000000000 | 0x0000000000000016e | 0x000000000000000f
0x00000005 | 0x0000000007fed6000 | 0x0000000000000000 | 0x0000000000000030 | 0x800000000000000f
0x00000006 | 0x0000000007ff06000 | 0x0000000000000000 | 0x0000000000000024 | 0x800000000000000f
0x00000000 | 0x0000000007ff2a000 | 0x0000000000000000 | 0x0000000000000004 | 0x000000000000000f
0x00000009 | 0x0000000007ff2e000 | 0x0000000000000000 | 0x0000000000000008 | 0x000000000000000f
0x0000000a | 0x0000000007ff36000 | 0x0000000000000000 | 0x0000000000000004 | 0x000000000000000f
0x00000004 | 0x0000000007ff3a000 | 0x0000000000000000 | 0x0000000000000086 | 0x000000000000000f
0x00000006 | 0x0000000007ffc0000 | 0x0000000000000000 | 0x0000000000000020 | 0x800000000000000f
0x00000007 | 0x0000000007ffe0000 | 0x0000000000000000 | 0x0000000000000010 | 0x000000000000000f
PS1:\efi\boot\> _
```

Next Steps

A good UEFI assembly language introduction can be found at:

<http://x86asm.net/articles/uefi-programming-first-steps/index.html>

However, with what we've shown above and the enhanced functionality available to UASM you can leverage any C/C++ UEFI based tutorials with minimal to no translation required!

We have successfully tested and used a number of the core protocols already including networking, GOP, File Systems, Pointers and Console input and output.

For a reference to all available UEFI Shell commands and options:

http://h17007.www1.hp.com/docs/iss/proliant_uefi/UEFI_TM_030617/GUID-D7147C7F-2016-0901-0A6D-00000000E1B.html