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

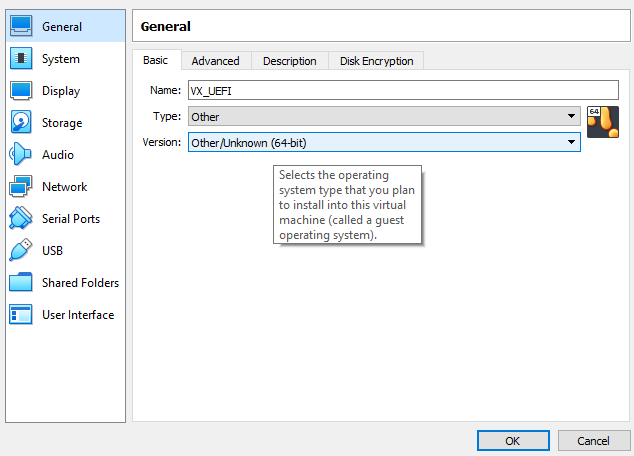
Format the USB Flash drive as FAT32. Create a folder structure \EFI\BOOT\ on the driver. This is the default location for the firmware to look for the initial EFI application to load. The application / depending on your architecture will be named something like BOOTX64.EFI. For our example we will assume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

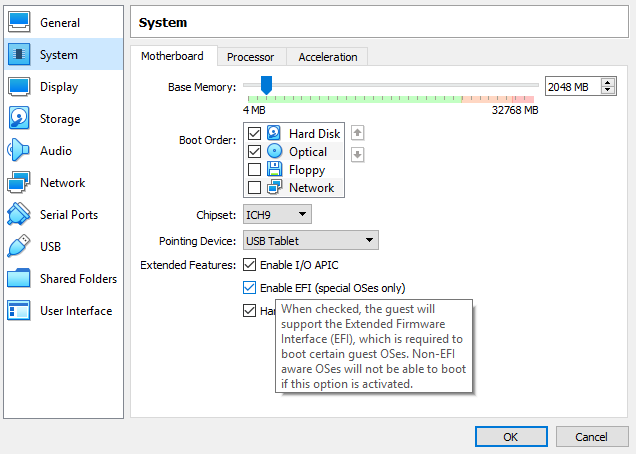
To the /EFI/BOOT/ folder you created earlier on the USB stick. Rename the file to **BOOTX64.EFI.**

# Configure the VM

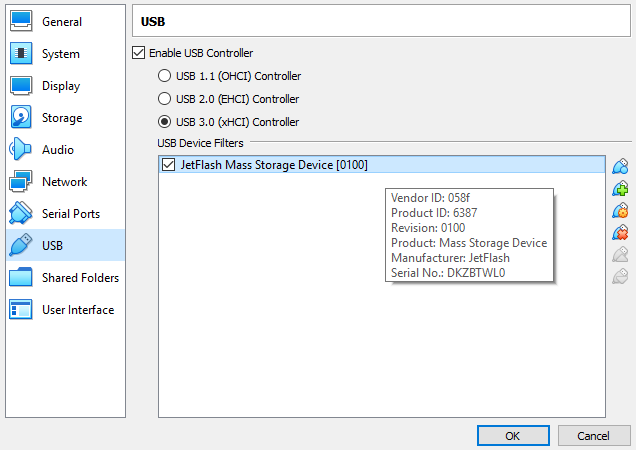
Ensure that the VM is configured as 64bit other:

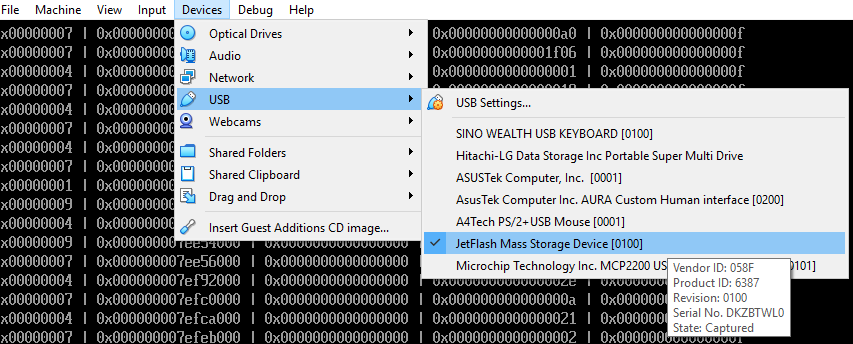


Ensure the VM is configured to use EFI:



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 provider 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

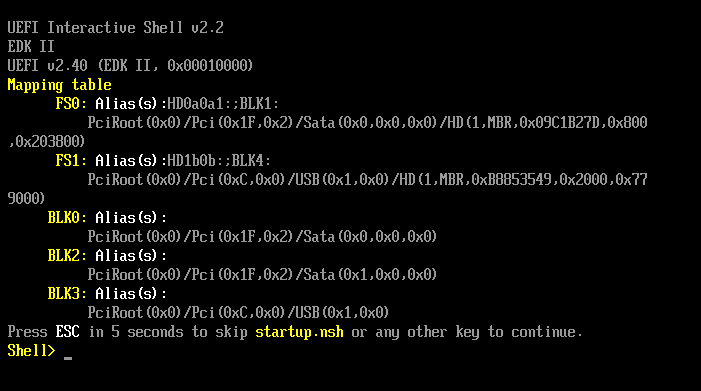
Main ENDP

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*

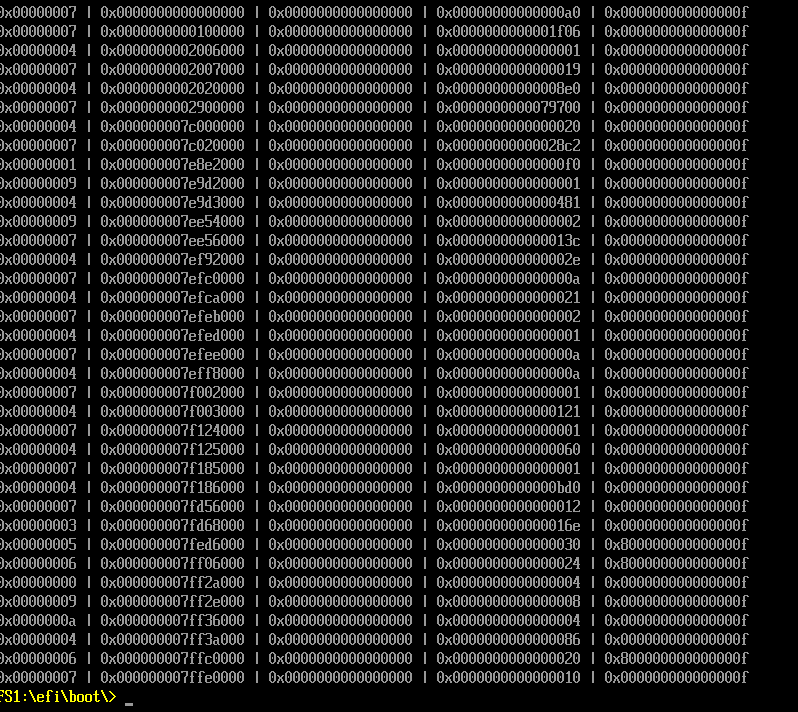
# Running the Example



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:



# 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.hpe.com/docs/iss/proliant_uefi/UEFI_TM_030617/GUID-D7147C7F-2016-0901-0A6D-000000000E1B.html>