

UEFI & EDK II TRAINING

How to Write a UEFI Driver Lab

tianocore.org

LESSON OBJECTIVE

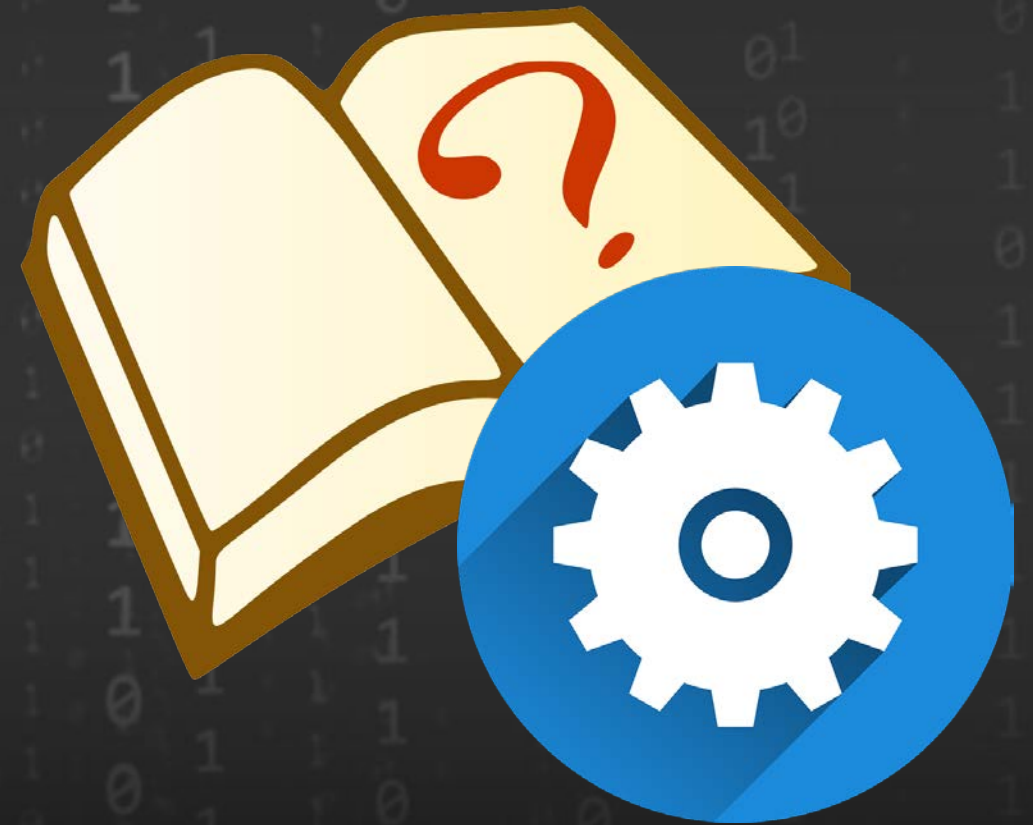
- ★ Compile a UEFI driver template created from UEFI Driver Wizard
- ★ Test driver in QEMU using UEFI Shell 2.0
- ★ Port code into the template driver

UEFI DRIVER PORTING LAB

This Lab uses the template UEFI driver created by the UEFI Driver Wizard

Lab 2: Building a UEFI Driver

In this lab, you'll build a UEFI Driver created by the UEFI Driver Wizard. You will include the driver in the OVMF project. Build the UEFI Driver from the Driver Wizard



Compile a UEFI Driver

Two Ways to Compile a Driver	
<i>Standalone</i>	<i>In a Project</i>
The build command directly compiles the .INF file	Include the .INF file in the project's .DSC file
Results: The driver's .EFI file is located in the Build directory	Results: The driver's .EFI file is a part of the project in the Build directory

Lab 2: Build the UEFI Driver

- Perform Lab Setup from previous Labs
- Open `~src/edk2/OvmfPkg/OvmfPkgX64.dsc`
- Add the following to the [Components] section:

Hint: add to the last module in the [Components] section

```
# Add new modules here
  MyWizardDriver/MyWizardDriver.inf{
    <LibraryClasses>      DebugLib|MdePkg/Library/UefiDebugLibConOut/UefiDebugLibConOut.inf
  }
```

- Save and close the file `~src/edk2/OvmfPkg/OvmfPkgX64.dsc`

Lab 2: Build and Test Driver

Build MyWizardDriver – Cd to ~/src/edk2 dir

```
bash$ . edksetup.sh  
bash$ build
```

Build error Known issue from UEFI Driver Wizard:
ComponentName.c Line 148 col 74 needs “//” in front of “## TO_START”

```
bash$ build
```

Build ERRORS: Copy the solution files from
~/FW/LabSampleCode/LabSolutions/LessonC.1 to ~/src/edk2/MyWizardDriver

Lab 2: Build and Test Driver

Copy MyWizardDriver.efi to hda-content

```
bash$ cd ~/run-ovmf/hda-content
bash$ cp ~/src/edk2/Build/OvmfX64/DEBUG_GCC5/X64/MyWizardDriver.efi .
```

Test by Invoking Qemu

```
bash$ cd ~/run-ovmf
bash$ . RunQemu.sh
```

Load the UEFI Driver from the shell

At the Shell prompt, type fs0:

Type: load MyWizardDriver.efi

```
Shell> fs0:
FS0:\> load MyWizardDriver.efi
Image 'FS0:\MyWizardDriver.efi' loaded at 5E7F000 - Success
FS0:\> _
```


Lab 2: Test Driver

At the shell prompt Type: drivers

Verify the UEFI Shell loaded the new driver. The drivers command will display the driver information and a driver handle number ("a9" in the example screenshot)

```
92 00000011 ? - - - - Usb Mass Storage Driver      UsbMassStorageDxe
93 00000010 B - - 1 1 QEMU Video Driver           QemuVideoDxe
94 00000010 ? - - - - Virtio GPU Driver          VirtioGpuDxe
A9 00000000 ? - - - - MyWizardDriver             \MyWizardDriver.efi
FS0:\>
```


Lab 2: Test Driver

At the shell prompt using the handle from the drivers command,
Type: `dh -d a9`

Note: The value a9 is the driver handle for MyWizardDriver. The handle value may change based on your system configuration.(see example screenshot - right)

```
FS0:\> dh -d a9
A9: SupportedEfiSpecVersion(0x0002003C) ComponentName2 ComponentName DriverBin
ng HiiPackageList ImageDevicePath(..0xFBFC1)\MyWizardDriver.efi) LoadedImage(
yWizardDriver.efi)
  Driver Name [A9]      : MyWizardDriver
  Driver Image Name    : \MyWizardDriver.efi
  Driver Version       : 00000000
  Driver Type          : <Unknown>
  Configuration        : NO
  Diagnostics          : NO
  Managing              : None
FS0:\> _
```


Lab 2: Test Driver

At the shell prompt using the handle from the drivers command,
Type: `unload a9`

See example screenshot - below
Type: `drivers` again

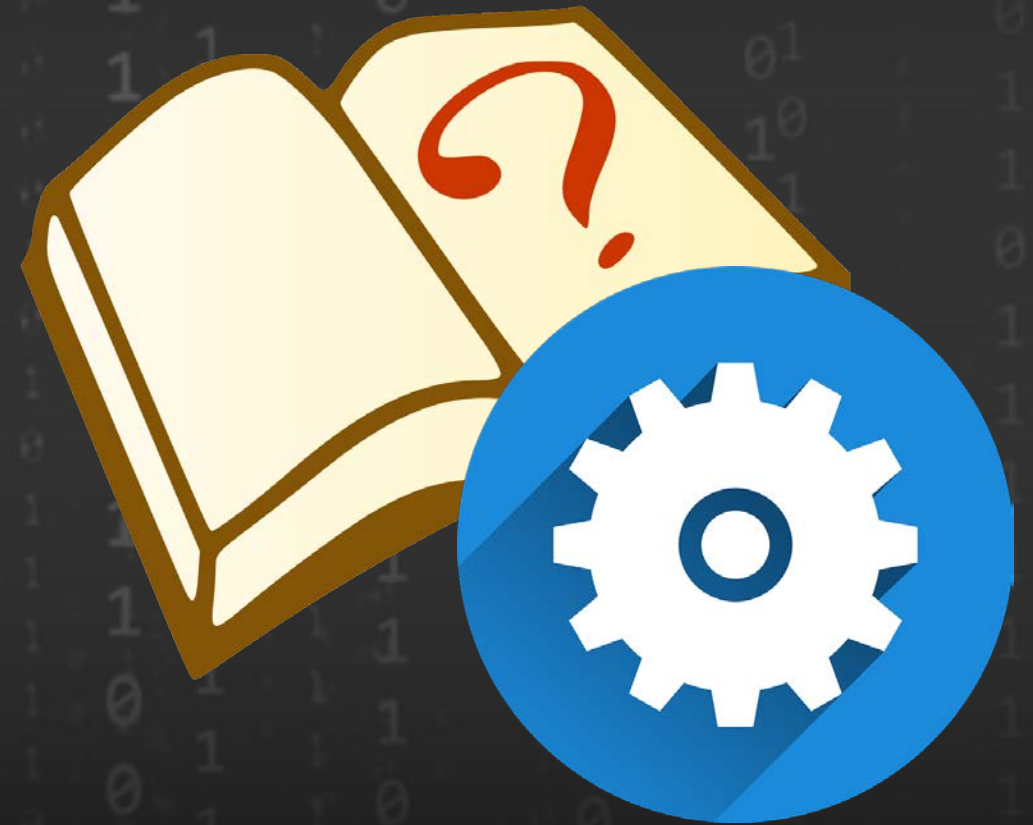
Notice results of `unload` command

Exit QEMU

```
FS0:\> unload a9
Unload - Handle [6B1B798]. [y/n]?
y
Unload - Handle [6B1B798] Result Success.
FS0:\> _
```

Lab 3: Component Name

In this lab, you'll change the information reported to the drivers command using the `ComponentName` and `ComponentName2` protocols.



Lab 3: Component Name

- Open `~/src/edk2/MyWizardDriver/ComponentName.c`
- Change the string returned by the driver from MyWizardDriver to: UEFI Sample Driver

```
/// Table of driver names
///
GLOBAL_REMOVE_IF_UNREFERENCED
EFI_UNICODE_STRING_TABLE mMyWizardDriverDriverNameTable[] = {
    { "eng;en", (CHAR16 *)L"UEFI Sample Driver" },
    { NULL, NULL }
};
```

- Save and close the file: `~/src/edk2/MyWizardDriver/ComponentName.c`

Lab 3: Build and Test Driver

Build MyWizardDriver – Cd to ~/src/edk2 dir

```
bash$ build
```

Copy MyWizardDriver.efi to hda-content

```
bash$ cd ~/run-ovmf/hda-content
```

```
bash$ cp ~/src/edk2/Build/OvmfX64/DEBUG_GCC5/X64/MyWizardDriver.efi .
```

Test by Invoking Qemu

```
bash$ cd ~/run-ovmf
```

```
bash$ . RunQemu.sh
```


Lab 3: Build and Test Driver

Load the UEFI Driver from the shell

At the Shell prompt, type `Shell> fs0:`

Type: `FS0:\> load MyWizardDriver.efi`

Type: `drivers`

Observe the change in the string
that the driver returned

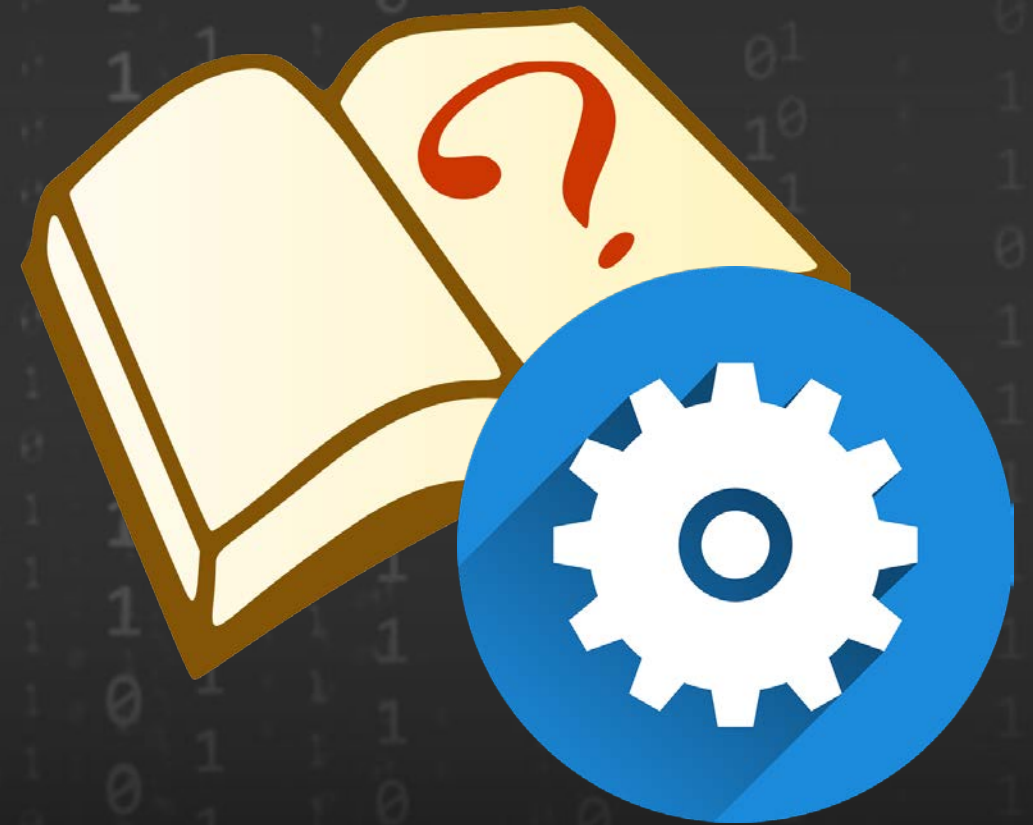
Exit QEMU

```
92 00000011 ? - - - - Usb Mass Storage Driver      UsbMassStorageDxe
93 00000010 B - - 1 1 QEMU Video Driver             QemuVideoDxe
94 00000010 ? - - - - Virtio GPU Driver             VirtioGpuDxe
A9 00000000 ? - - - - - UEFI Sample Driver       \MyWizardDriver.efi
FS0:\> _
```

Lab 4: Porting the Supported & Start Functions

The UEFI Driver Wizard produced a starting point for driver porting ... so now what?

In this lab, you'll port the "Supported" and "Start" functions for the UEFI driver



Lab 4: Porting Supported and Start



Review the Driver Binding Protocol



Supported()

Determines if a driver supports a controller



Start()

Starts a driver on a controller & Installs Protocols



Stop()

Stops a driver from managing a controller

Lab 4: The Supported() Port

The UEFI Driver Wizard produced a Supported() function but it only returns `EFI_UNSUPPORTED`

Supported Goals:

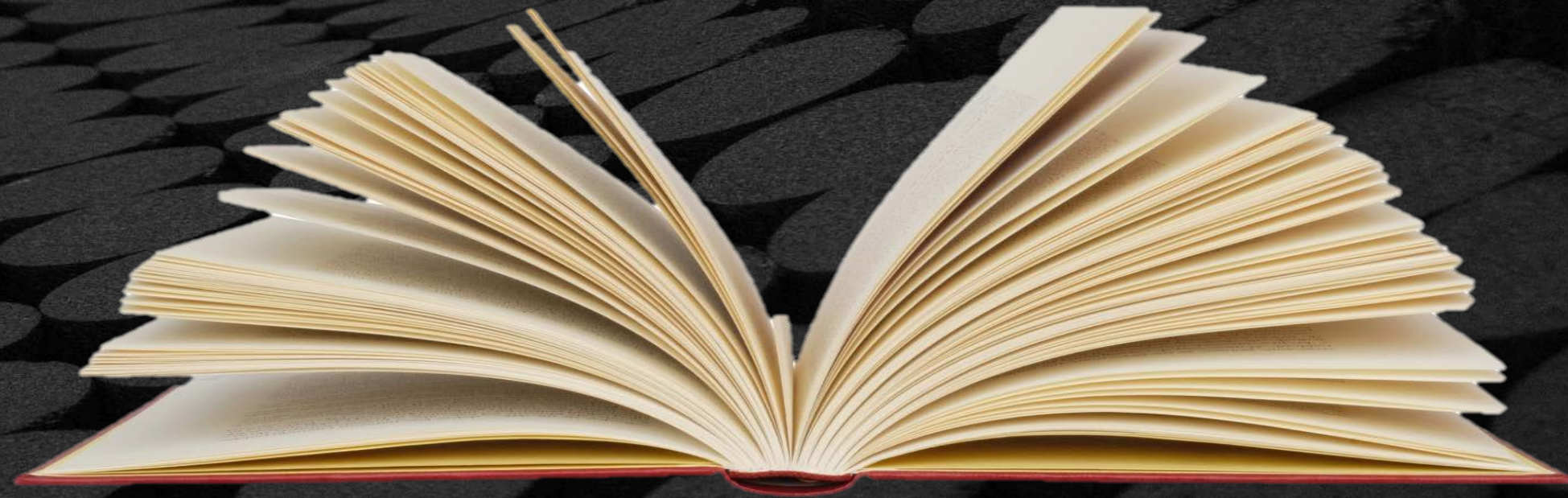
- Checks if the driver supports the device for the specified controller handle
- Associates the driver with the Serial I/O protocol
- Helps locate a protocol's specific GUID through UEFI Boot Services' function

Lab 4: Help from Robust Libraries

EDK II has libraries to help with porting UEFI Drivers

 AllocateZeroPool() include - [MemoryAllocationLib.h]

 SetMem16() include - [BaseMemoryLib.h]



Check the MdePkg with libraries help file (.chm format)

Lab 4: Update Supported

- Open `~/src/edk2/MyWizardDriver/MyWizardDriver.c`
- Locate `MyWizardDriverDriverBindingSupported()`, the supported function for this driver and comment out the `"//"` in the line: `"return EFI_UNSUPPORTED; "`

```
EFI_STATUS
EFIAPI
MyWizardDriverDriverBindingSupported (
    IN EFI_DRIVER_BINDING_PROTOCOL *This,
    IN EFI_HANDLE                  ControllerHandle,
    IN EFI_DEVICE_PATH_PROTOCOL    *RemainingDevicePath OPTIONAL
)
{
    // return EFI_UNSUPPORTED;
}
```

- copy and past (next slide)

Lab 4: Update Supported Add Code

Copy & Paste the following code for the supported function
`MyWizardDriverDriverBindingSupported()`:

```
EFI_STATUS Status;  
EFI_SERIAL_IO_PROTOCOL *SerialIo;  
Status = gBS->OpenProtocol (  
    ControllerHandle,  
    &gEfiSerialIoProtocolGuid,  
    (VOID **) &SerialIo,  
    This->DriverBindingHandle,  
    ControllerHandle,  
    EFI_OPEN_PROTOCOL_BY_DRIVER | EFI_OPEN_PROTOCOL_EXCLUSIVE  
);  
  
if (EFI_ERROR (Status)) {  
    return Status; // Bail out if OpenProtocol returns an error  
}  
  
// We're here because OpenProtocol was a success, so clean up  
gBS->CloseProtocol (  
    ControllerHandle,  
    &gEfiSerialIoProtocolGuid,  
    This->DriverBindingHandle,  
    ControllerHandle  
);  
  
return EFI_SUCCESS;
```


Lab 4: Notice UEFI Driver Wizard Includes

- Open `~/src/edk2/MyWizardDriver/MyWizardDriver.h`
- Notice the following include statement is already added by the driver wizard:

```
// Produced Protocols
//
#include <Protocol/SerialIo.h>
```

- Review the Libraries section and see that UEFI Driver Wizard automatically includes library headers based on the form information. Also other common library headers were included

```
// Libraries
//
#include <Library/UefiBootServicesTableLib.h>
#include <Library/MemoryAllocationLib.h>
#include <Library/BaseMemoryLib.h>
#include <Library/BaseLib.h>
#include <Library/UefiLib.h>
#include <Library/DevicePathLib.h>
#include <Library/DebugLib.h>
```


Lab 4: Update the Start()

- **Copy & Paste** the following in MyWizardDriver.c after the #include "MyWizardDriver.h" line:

```
#define DUMMY_SIZE 100*16 // Dummy buffer
CHAR16 *DummyBufferfromStart = NULL;
```

Locate MyWizardDriverDriverBindingStart(), the start function for this driver and comment out the "//" in the line "return EFI_UNSUPPORTED; "

```
EFI_STATUS
EFIAPI
MyWizardDriverDriverBindingStart (
    IN EFI_DRIVER_BINDING_PROTOCOL *This,
    IN EFI_HANDLE ControllerHandle,
    IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath OPTIONAL
)
{
    // return EFI_UNSUPPORTED;
}
```


Lab 4: Update Start Add Code

Copy & Paste the following code for the start function

MyWizardDriverDriverBindingStart():

```
if (DummyBufferfromStart == NULL) {      // was buffer already allocated?
    DummyBufferfromStart = (CHAR16*)AllocateZeroPool (DUMMY_SIZE * sizeof(CHAR16));
}

if (DummyBufferfromStart == NULL) {
    return EFI_OUT_OF_RESOURCES;    // Exit if the buffer isn't there
}

SetMem16 (DummyBufferfromStart, (DUMMY_SIZE * sizeof(CHAR16)), 0x0042); // Fill buffer

return EFI_SUCCESS;
```

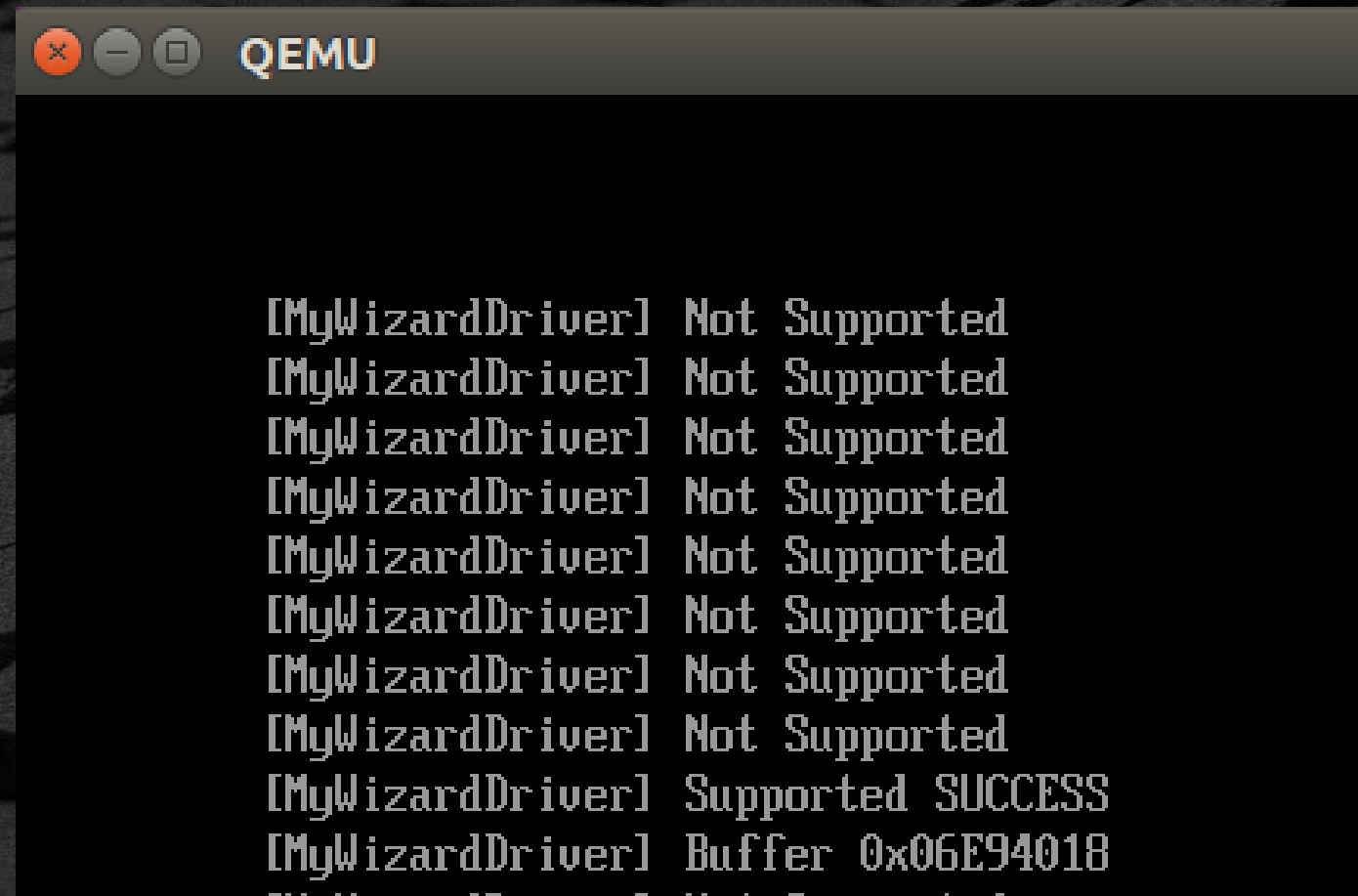
- Notice the Library calls to `AllocateZeroPool()` and `SetMem16()`
- The `Start()` function is where there would be calls to `"gBS-InstallMultipleProtocolInterfaces()"`

Lab 4: Debugging before Testing the Driver

UEFI drivers can use the EDK II debug library

 `DEBUG()` include - [DebugLib.h]

`DEBUG()` Macro statements can show status progress interest points throughout the driver code



```
QEMU

[MyWizardDriver] Not Supported
[MyWizardDriver] Not Supported
[MyWizardDriver] Not Supported
[MyWizardDriver] Not Supported
[MyWizardDriver] Not Supported
[MyWizardDriver] Not Supported
[MyWizardDriver] Not Supported
[MyWizardDriver] Not Supported
[MyWizardDriver] Not Supported
[MyWizardDriver] Supported SUCCESS
[MyWizardDriver] Buffer 0x06E94018
```


Lab 4: Add Debug Statements Supported()

Copy & Paste the following DEBUG() macros for the supported function:

```
Status = gBS->OpenProtocol(
    ControllerHandle,
    &gEfiSerialIoProtocolGuid,
    (VOID **)&SerialIo,
    This->DriverBindingHandle,
    ControllerHandle,
    EFI_OPEN_PROTOCOL_BY_DRIVER | EFI_OPEN_PROTOCOL_EXCLUSIVE
);

if (EFI_ERROR(Status)) {
    DEBUG((EFI_D_INFO, "[MyWizardDriver] Not Supported \r\n"));
    return Status; // Bail out if OpenProtocol returns an error
}

// We're here because OpenProtocol was a success, so clean up
gBS->CloseProtocol(
    ControllerHandle,
    &gEfiSerialIoProtocolGuid,
    This->DriverBindingHandle,
    ControllerHandle
);
DEBUG((EFI_D_INFO, "[MyWizardDriver] Supported SUCCESS\r\n"));
return EFI_SUCCESS;
```


Lab 4: Add Debug Statements Start()

Copy & Paste the following DEBUG macro for the Start function just before the `return EFI_SUCCESS;` statement

```
DEBUG ((EFI_D_INFO, "\r\n***\r\n[MyWizardDriver] Buffer 0x%08x\r\n", DummyBufferfromStart));  
return EFI_SUCCESS;
```

Note: This debug macro displays the memory address of the allocated buffer on the debug console

Save `~/src/edk2/MyWizardDriver/MyWizardDriver.c`

Lab 4: Build and Test Driver

Build MyWizardDriver – Cd to ~/src/edk2 dir

```
bash$ build
```

Copy MyWizardDriver.efi to hda-content

```
bash$ cd ~/run-ovmf/hda-content
```

```
bash$ cp ~/src/edk2/Build/OvmfX64/DEBUG_GCC5/X64/MyWizardDriver.efi .
```

Test by Invoking Qemu

```
bash$ cd ~/run-ovmf
```

```
bash$ . RunQemu.sh
```


Lab 4: Build and Test Driver

Load the UEFI Driver from the shell

At the Shell prompt, type **Shell> fs0:**

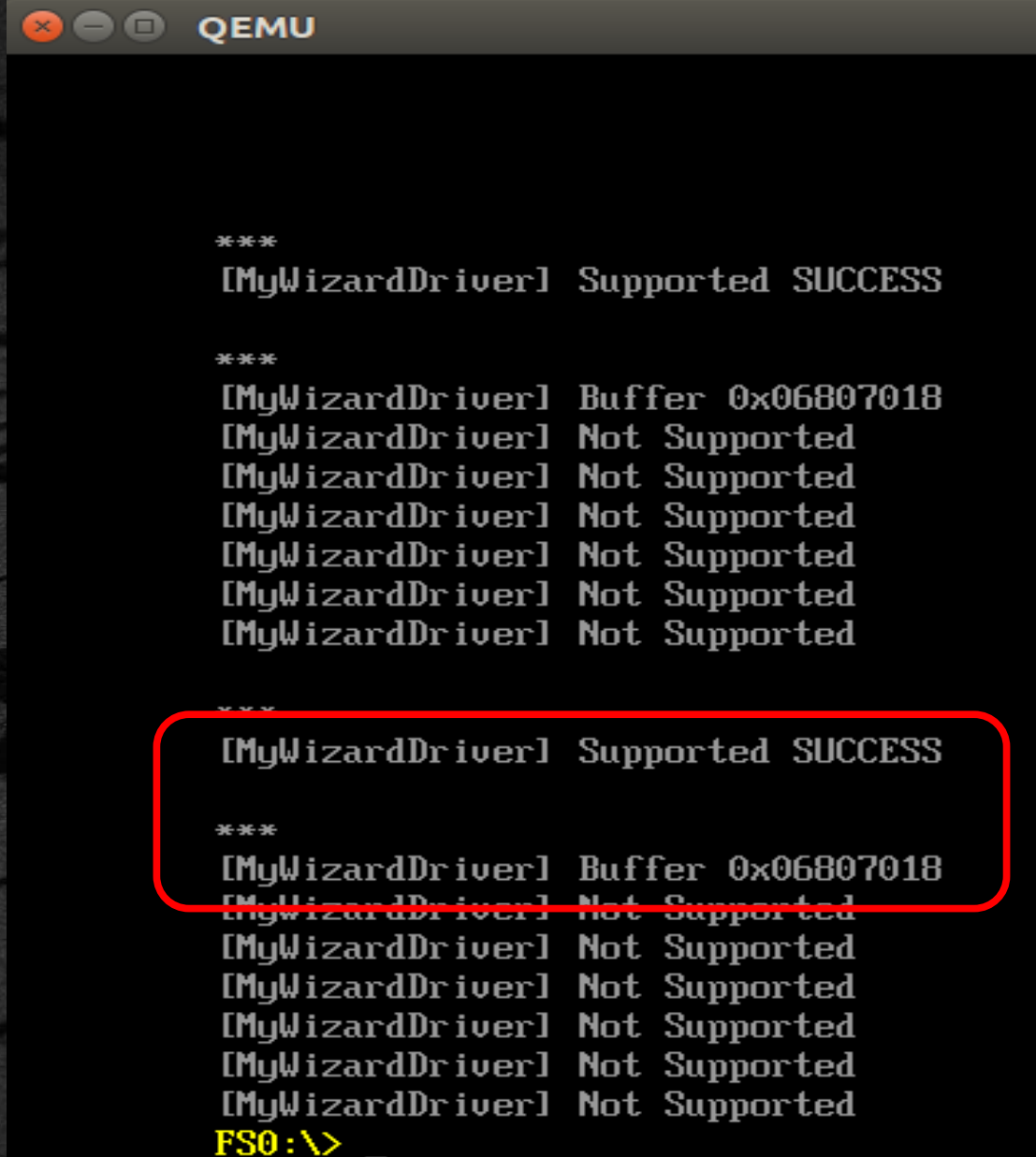
Type: **FS0:\> load MyWizardDriver.efi**

```
Shell> fs0:
FS0:\> load MyWizardDriver.efi
Image 'FS0:\MyWizardDriver.efi' loaded at 5E7F000 - Success
FS0:\> _
```


Lab 4: Build and Test Driver

- Check the QEMU debug console output.
- Notice Debug messages indicate the driver did not return `EFI_SUCCESS` from the “Supported()” function most of the time.
- See that the “Start()” function did get called and a Buffer was allocated.

Exit QEMU



```
QEMU

***
[MyWizardDriver] Supported SUCCESS

***
[MyWizardDriver] Buffer 0x06807018
[MyWizardDriver] Not Supported
[MyWizardDriver] Not Supported
[MyWizardDriver] Not Supported
[MyWizardDriver] Not Supported
[MyWizardDriver] Not Supported
[MyWizardDriver] Not Supported

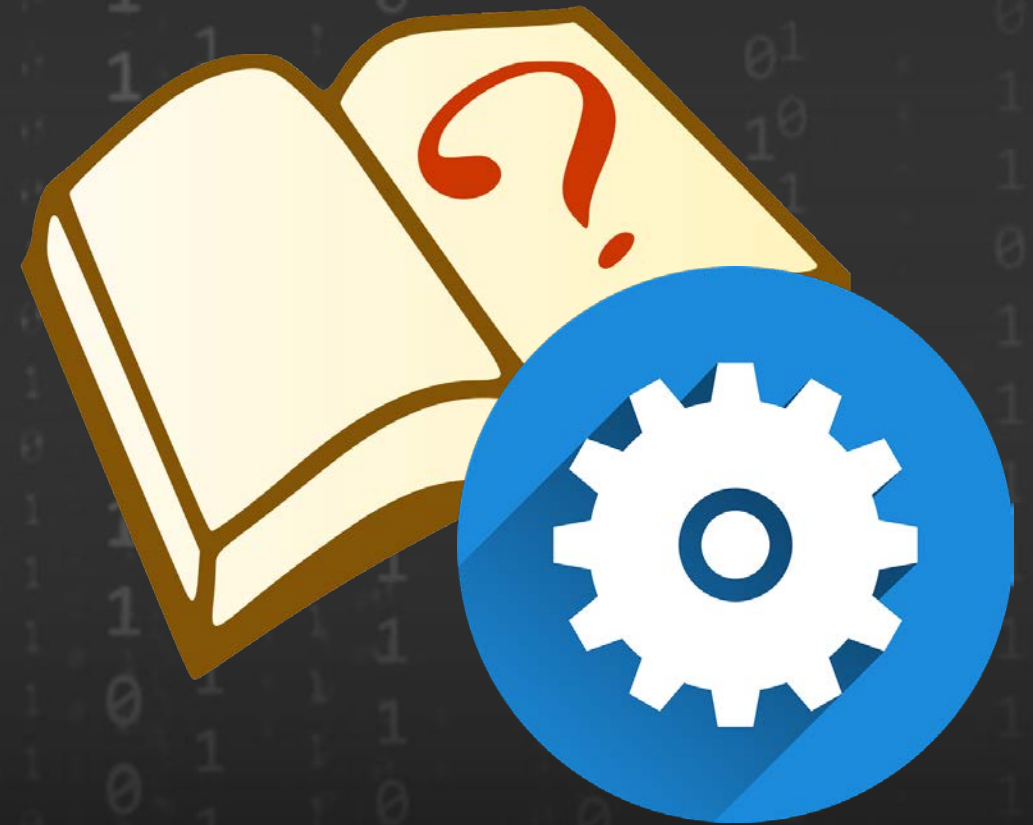
***
[MyWizardDriver] Supported SUCCESS

***
[MyWizardDriver] Buffer 0x06807018
[MyWizardDriver] Not Supported
[MyWizardDriver] Not Supported
[MyWizardDriver] Not Supported
[MyWizardDriver] Not Supported
[MyWizardDriver] Not Supported
[MyWizardDriver] Not Supported
FS0:\> _
```


Lab 5: Create a NVRAM Variable

In this lab you'll create a non-volatile UEFI variable (NVRAM), and set and get the variable to return a successful supported function

Use Runtime services to
"SetVariable()" and
"GetVariable()"



Lab 5: Adding a NVRAM Variable Steps

1. Create .h file with new typedef definition and its own GUID
2. Include the new .h file in the driver's top .h file
3. EntryPoint() Init new buffer for NVRam Variable
4. Supported() make a call to a new function to set/get the new NVRam Variable
5. Before EntryPoint() add the new function CreateNVVariable() to the driver.c file.

Lab 5: Create a new .h file

Create a new file in your editor called: "MyWizardDriverNVDataStruc.h"

Copy, Paste and then **Save** this file

```
#ifndef _MYWIZARDDRIVERNVDATASTRUC_H_
#define _MYWIZARDDRIVERNVDATASTRUC_H_
#include <Guid/HiiPlatformSetupFormset.h>
#include <Guid/HiiFormMapMethodGuid.h>

#define MYWIZARDDRIVER_VAR_GUID \
{ \
    0x363729f9, 0x35fc, 0x40a6, 0xaf, 0xc8, 0xe8, 0xf5, 0x49, 0x11, 0xf1, 0xd6 \
}

#pragma pack(1)
typedef struct {

    UINT16    MyWizardDriverStringData[20];
    UINT8     MyWizardDriverHexData;
    UINT8     MyWizardDriverBaseAddress;
    UINT8     MyWizardDriverChooseToEnable;

} MYWIZARDDRIVER_CONFIGURATION;

#pragma pack()
#endif
```


Lab 5: Update MyWizardDriver.c

Open `"~/src/edk2/MyWizardDriver/MyWizardDriver.c"`

Copy & Paste the following 4 lines after the `#include "MyWizardDriver.h"` statement:

```
#include "MyWizardDriver.h"

EFI_GUID    mMyWizardDriverVarGuid = MYWIZARDDRIVER_VAR_GUID;

CHAR16      mVariableName[] = L"MWD_NVData"; // Use Shell "Dmpstore" to see
MYWIZARDDRIVER_CONFIGURATION mMyWizDrv_Conf_buffer;
MYWIZARDDRIVER_CONFIGURATION *mMyWizDrv_Conf = &mMyWizDrv_Conf_buffer; //use the pointer
```


Lab 5: Update MyWizardDriver.c

Locate "MyWizardDriverDriverBindingSupported ()" function

Comment out the DEBUG macro statement and return statement as below:

Copy & Paste the 5 lines: 1) new call to "CreateNVVariable();" , 2-4) if statement and 5) "return" as below:

```
if (EFI_ERROR(Status)) {  
    //DEBUG((EFI_D_INFO, "[MyWizardDriver] Not Supported \r\n"));  
    //return Status; // Bail out if OpenProtocol returns an error  
    Status = CreateNVVariable();  
    if (EFI_ERROR(Status)) {  
        DEBUG((EFI_D_ERROR, "[MyWizardDriver] Not Supported \r\n"));  
    }  
    return Status; // Status now depends on CreateNVVariable Function
```


Lab 5: Update MyWizardDriver.c

Copy & Paste the new function before the call to "MyWizardDriverDriverEntryPoint()"

```
EFI_STATUS
EFI_API
CreateNVVariable()
{
    EFI_STATUS      Status;
    UINTN           BufferSize;

    BufferSize = sizeof (MYWIZARDDRIVER_CONFIGURATION);
    Status = gRT->GetVariable(
        mVariableName,
        &mMyWizardDriverVarGuid,
        NULL,
        &BufferSize,
        mMyWizDrv_Conf
    );
    if (EFI_ERROR(Status)) { // Not defined yet so add it to the NV Variables.
        if (Status == EFI_NOT_FOUND) {
            Status = gRT->SetVariable(
                mVariableName,
                &mMyWizardDriverVarGuid,
                EFI_VARIABLE_NON_VOLATILE | EFI_VARIABLE_BOOTSERVICE_ACCESS,
                sizeof (MYWIZARDDRIVER_CONFIGURATION),
                mMyWizDrv_Conf // buffer is 000000 now for first time set
            );
            DEBUG((EFI_D_INFO, "[MyWizardDriver] Variable %s created in NVRam Var\r\n", mVariableName));
            return EFI_SUCCESS;
        }
    }
    // already defined once
    return EFI_UNSUPPORTED;
}
```


Lab 5: Update MyWizardDriver.h

Open "~/src/edk2/MyWizardDriver/MyWizardDriver.h"

Copy & Paste the following "#include" after the list of library include statements:

```
// Libraries
// . . .
#include <Library/UefiRuntimeServicesTableLib.h>
```

Copy & Paste the following "#include" after the list of protocol include statements:

```
// Produced Protocols
// . . .
#include "MyWizardDriverNVDataStruc.h"
```

Save "~/src/edk2/MyWizardDriver/MyWizardDriver.h"

Save "~/src/edk2/MyWizardDriver/MyWizardDriver.c"

Lab 5: Build and Test Driver

Build MyWizardDriver – Cd to ~/src/edk2 dir

```
bash$ build
```

Copy MyWizardDriver.efi to hda-content

```
bash$ cd ~/run-ovmf/hda-content
```

```
bash$ cp ~/src/edk2/Build/OvmfX64/DEBUG_GCC5/X64/MyWizardDriver.efi .
```

Test by Invoking Qemu

```
bash$ cd ~/run-ovmf
```

```
bash$ . RunQemu.sh
```


Lab 5: Test Driver

Load the UEFI Driver from the shell

At the Shell prompt, type `Shell> fs0:`

Type: `FS0:\> load MyWizardDriver.efi`

Observe the Buffer address
returned by the debug statement

```
Shell> fs0:
FS0:\> load MyWizardDriver.efi
Image 'FS0:\MyWizardDriver.efi' loaded at 6801000 - Success
[MyWizardDriver] Supported SUCCESS with Faux Supported by NVRam Var
```

```
***
```

```
[MyWizardDriver] Buffer 0x06808018
```


Lab 5: Verify Driver

At the Shell prompt, type **FS0:\> mem 0x6808018**

Observe the Buffer is filled with the letter "B" or 0x0042

```
[MyWizardDriver] Buffer 0x06808018
```

```
FS0:\>
```

```
FS0:\> mem 0x6808018
```

```
Memory Address 0000000006808018 200 Bytes
```

```
06808018: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00 *B.B.B.B.B.B.B.B.*
06808028: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00 *B.B.B.B.B.B.B.B.*
06808038: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00 *B.B.B.B.B.B.B.B.*
06808048: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00 *B.B.B.B.B.B.B.B.*
06808058: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00 *B.B.B.B.B.B.B.B.*
06808068: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00 *B.B.B.B.B.B.B.B.*
```


Lab 5: Verify NVRAM Created by Driver

At the Shell prompt, type `FS0:\> dmpstore -all -b`

Observe now the NVRAM variable "MWD_NVData" was created and filled with 0x00s

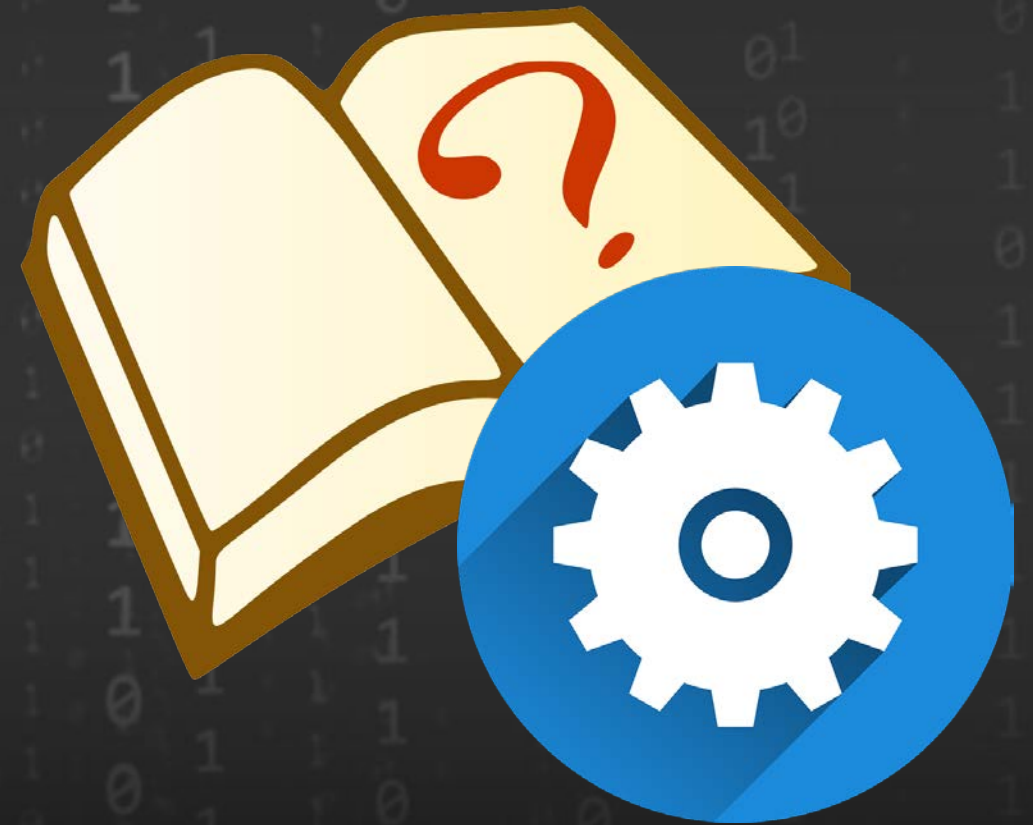
```
FS0:\> dmpstore -all -b
Variable NV+BS '363729F9-35FC-40A6-AFC8-E8F54911F1D6:MWD_NVData' DataSize = 0x2B

000000000: 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  *.....*
000000010: 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  *.....*
000000020: 00 00 00 00 00 00 00 00 00-00 00 00                    *.....*
```

Exit QEMU

Lab 6: Port Stop and Unload

In this lab, you'll port the driver's "Unload" and "Stop" functions to free any resources the driver allocated when it was loaded and started.



Lab 6: Port the Unload function

Open "~/src/edk2/MyWizardDriver/MyWizardDriver.c"

Locate "MyWizardDriverUnload ()" function

Copy & Paste the following "if" and "DEBUG" statements before the "return EFI_SUCCESS;" statement.

```
// Do any additional cleanup that is required for this driver
//
if (DummyBufferfromStart != NULL) {
    FreePool(DummyBufferfromStart);
    DEBUG((EFI_D_INFO, "[MyWizardDriver] Unload, clear buffer\r\n"));
}
DEBUG((EFI_D_INFO, "[MyWizardDriver] Unload success\r\n"));

return EFI_SUCCESS;
```


Lab 6: Port the Stop function

Locate "MyWizardDriverDriverBindingStop ()" function

Comment out with "//" before the "return EFI_UNSUPPORTED;" statement.

Copy & Paste the following "if" and "DEBUG" statements before the "return EFI_SUCCESS;" statement.

```
if (DummyBufferfromStart != NULL) {  
    FreePool(DummyBufferfromStart);  
    DEBUG((EFI_D_INFO, "[MyWizardDriver] Stop, clear buffer\r\n"));  
}  
DEBUG((EFI_D_INFO, "[MyWizardDriver] Stop, EFI_SUCCESS\r\n"));  
  
return EFI_SUCCESS;  
// return EFI_UNSUPPORTED;  
}
```

Save & Close "MyWizardDriverDriver.c"

Lab 6: Build and Test Driver

Build MyWizardDriver – Cd to ~/src/edk2 dir

```
bash$ build
```

Copy MyWizardDriver.efi to hda-content

```
bash$ cd ~/run-ovmf/hda-content
```

```
bash$ cp ~/src/edk2/Build/OvmfX64/DEBUG_GCC5/X64/MyWizardDriver.efi .
```

Test by Invoking Qemu

```
bash$ cd ~/run-ovmf
```

```
bash$ . RunQemu.sh
```


Lab 6: Test Driver

Load the UEFI Driver from the shell

At the Shell prompt, type `Shell> fs0:`

Type: `FS0:\> load MyWizardDriver.efi`

Observe the Buffer address is at
0x06808018 as this slide example

```
Shell> fs0:
FS0:\> load MyWizardDriver.efi
Image 'FS0:\MyWizardDriver.efi' loaded at 6801000 - Success
[MyWizardDriver] Supported SUCCESS with Faux Supported by NVRam Var
```

```
***
```

```
[MyWizardDriver] Buffer 0x06808018
```


Lab 6: Verify Driver

At the Shell prompt, type **FS0:\> drivers**

Observe the handle is "A9" as this slide example

Type: **mem 0x06808018**

Observe the buffer was filled with the "0x0042"

```
92 00000011 ? - - - - Usb Mass Storage Driver      UsbMassSt
93 00000010 B - - 1 1 QEMU Video Driver             QemuVideo
94 00000010 ? - - - - Virtio GPU Driver             VirtioGpu
A9 00000000 ? - - - - UEFI Sample Driver            \MyWizard
FS0:\> _
```

```
[MyWizardDriver] Buffer 0x06808018
```

```
FS0:\>
```

```
FS0:\> mem 0x6808018
```

```
Memory Address 0000000006808018 200 Bytes
```

```
06808018: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00 *B.B.
06808028: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00 *B.B.
06808038: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00 *B.B.
06808048: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00 *B.B.
06808058: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00 *B.B.
06808068: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00 *B.B.
```


Lab 6: Verify Unload

At the Shell prompt, type `FS0:\> unload a9`

Observe the DEBUG messages from the Unload

```
FS0:\> unload a9
Unload - Handle [6B1B798]. [y/n]?
y
Unload - Handle [6B1B798] Result Success.
FS0:\> _
```

```
[MyWizardDriver] Unload, clear buffer
[MyWizardDriver] Unload success
```


Lab 6: Verify Unload

At the Shell prompt, type `FS0:\> mem 0x06808018 -b`

Observe the buffer is now NOT filled

```
FS0:\> mem 6808018
Memory Address 0000000006808018 200 Bytes
06808018: AF AF AF AF AF AF AF AF-AF AF AF AF AF AF AF AF *.....*
06808028: AF AF AF AF AF AF AF AF-AF AF AF AF AF AF AF AF *.....*
06808038: AF AF AF AF AF AF AF AF-AF AF AF AF AF AF AF AF *.....*
06808048: AF AF AF AF AF AF AF AF-AF AF AF AF AF AF AF AF *.....*
```

Exit QEMU

Adding strings and forms to setup (HII)

Publish & consume protocols

Hardware initialization

Refer to the UEFI Drivers Writer's Guide for more tips— [Pdf link](#)

LESSON OBJECTIVE

- ★ Compile a UEFI driver template created from UEFI Driver Wizard
- ★ Test driver in QEMU using UEFI Shell 2.0
- ★ Port code into the template driver

Questions?

