tianocore

UEFI & EDK II TRAINING

How to Write a UEFI Driver - Porting Lab - Windows

tianocore.org

See also LabGuide.md for Copy & Paste examples in labs

Lesson Objective

- Compile a UEFI driver template created from UEFI Driver Wizard
- Test driver w/ Windows Emulation using UEFI Shell 2.0
- Port code in the template driver

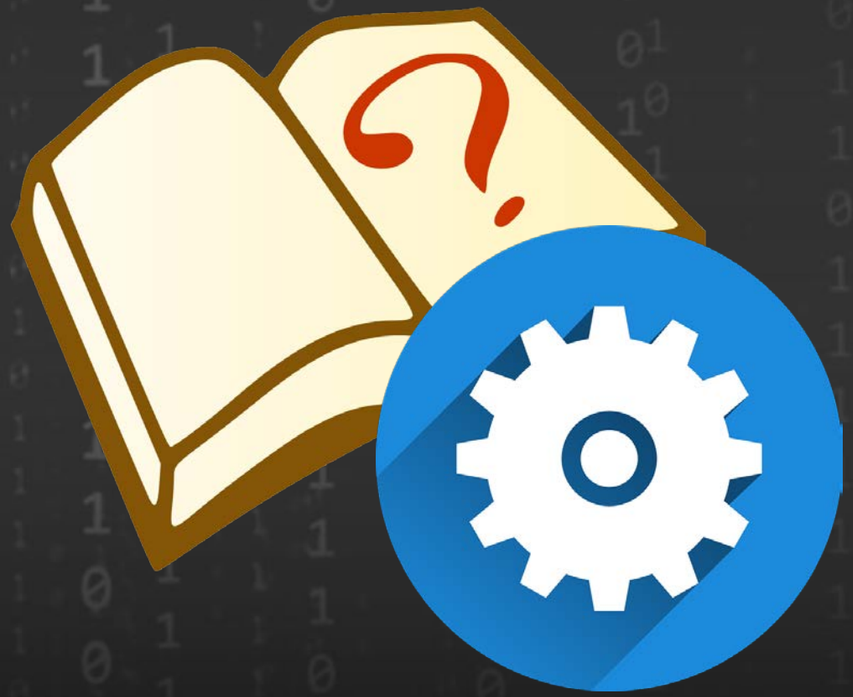
Note: Since this is a lab, to follow examples for copy & paste,
use the following Markdown link LabGuide.md

tianocore

LAB 1: UEFI DRIVER TEMPLATE

Use this lab, if you're not able to create a UEFI Driver Template using the UEFI Driver Wizard.

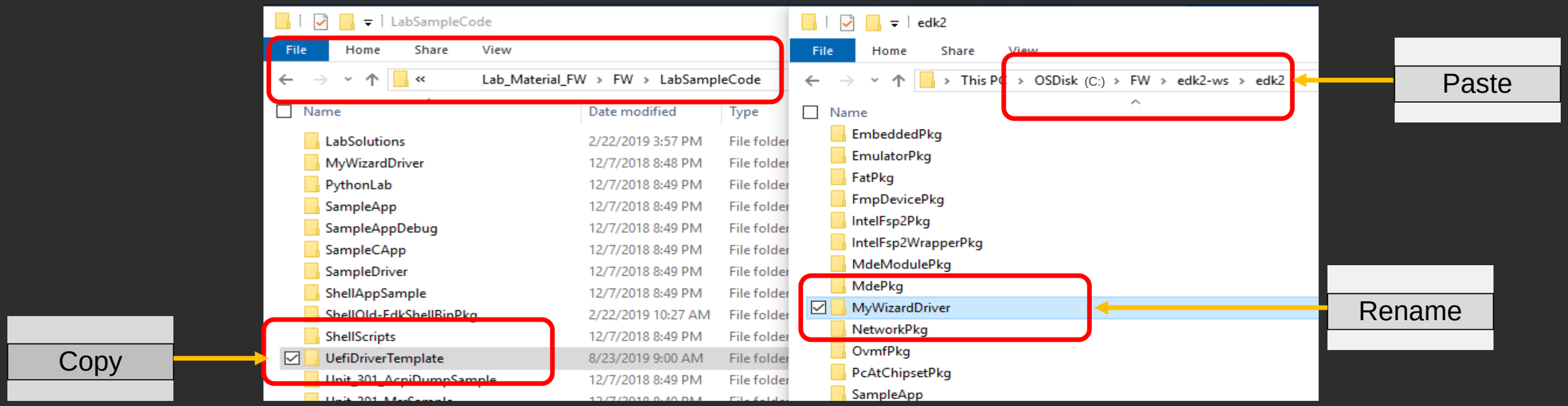
Note: Skip if LAB 1 UEFI Driver Wizard completed successfully



Lab 1: Get UEFI Driver Template

If UEFI Driver Wizard does not work:

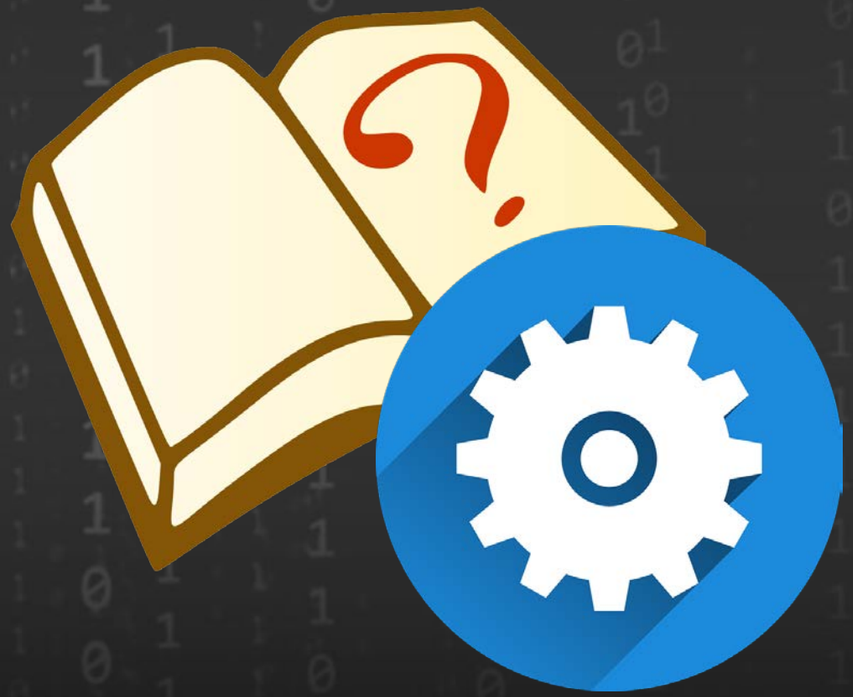
1. Copy the directory `UefiDriverTemplate` from
   `. . ./FW/LabSampleCode/ to C:/FW/edk2-ws/edk2`
2. Rename Directory `UefiDriverTemplate to MyWizardDriver`



Review [UEFI Driver Wizard Lab] for protocols produced and which are being consumed

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 Emulator project. Build the UEFI Driver from the Driver Wizard



Compile a UEFI Driver

Two Ways to Compile a Driver	
Standalone	*In a Project*
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 EmulatorPkg Labs
- Open `C:/FW/edk2-ws/edk2/EmulatorPkg/EmulatorPkg.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
```

- Save and close the file `C:/FW/edk2-ws/edk2/EmulatorPkg/EmulatorPkg.dsc`

Lab 2: Build and Test Driver

Open a VS Command Prompt and type: **`cd C:/FW/edk2-ws`** then

```
$> setenv.bat
$> cd edk2
$> edksetup
```

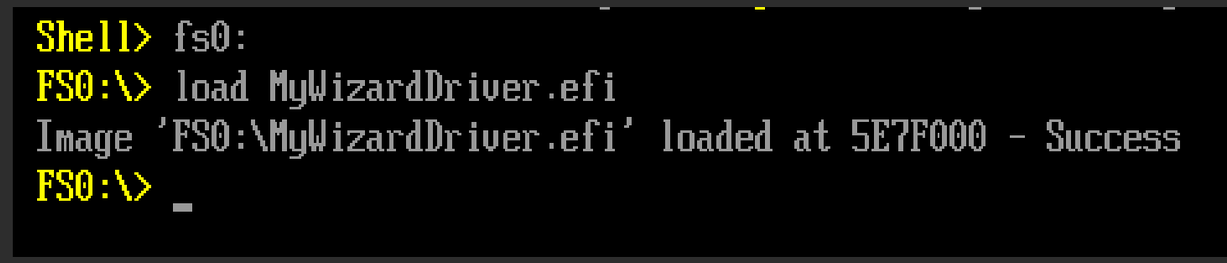
Build the MyWizardDriver with the Windows Emulation**

```
$> Build
$> RunEmulator.bat
```

Load the UEFI Driver from the shell

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

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



Build ERRORS: Copy the solution files from /FW/LabSampleCode/LabSolutions/LessonC.1 to C:/FW/edk2-ws/edk2/MyWizardDriver

** Make sure BUILD Switches from LAB C_03. . . are enabled in EmulatorPkg.dsc

Lab 2: Test Driver

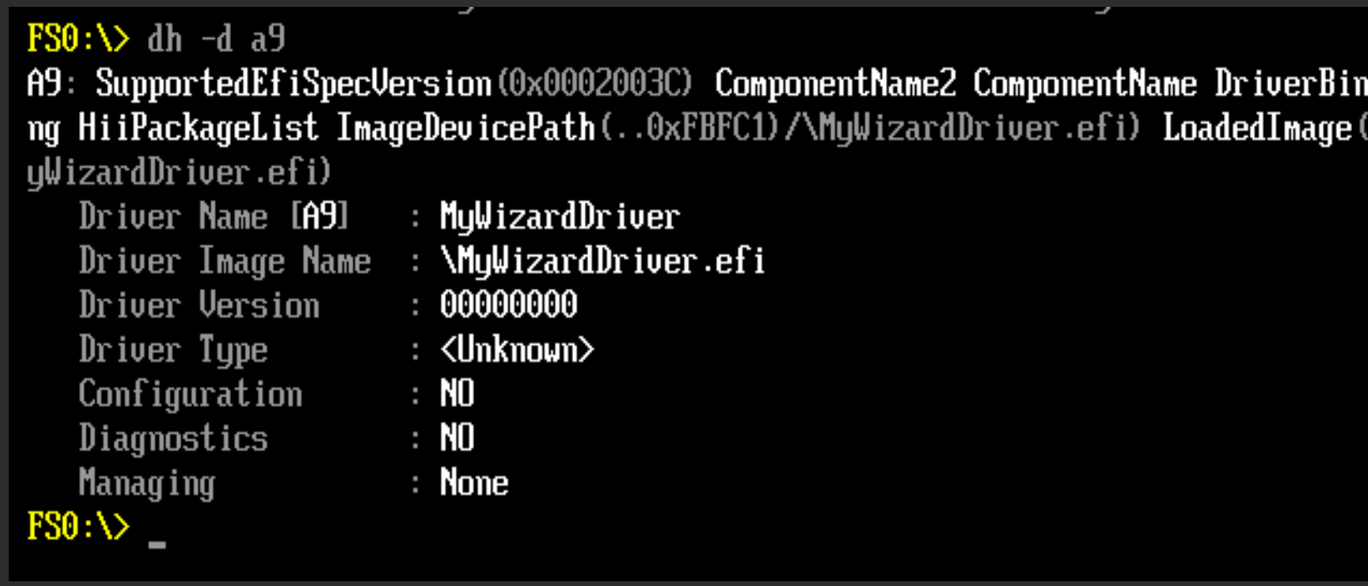
At the shell prompt Type: `FS0:\> drivers`

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)

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)



Lab 2: Test Driver

At the shell prompt using the handle from the drivers command,
Type: `FS0:/ > unload a9`

See example screenshot - below
Type: `drivers` again

Notice results of unload command

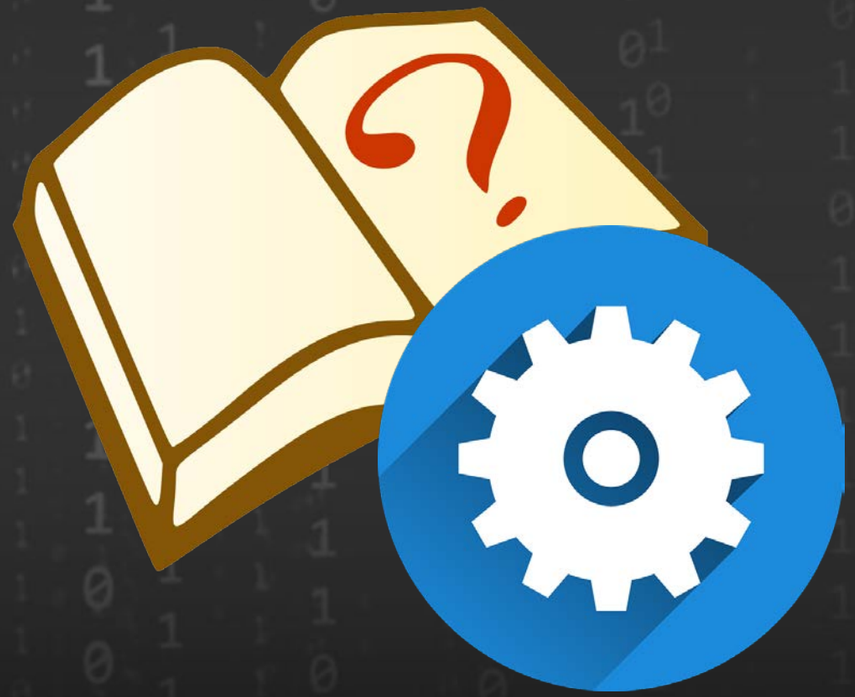
Exit type `FS0:/ > Reset`

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

tianocore

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** `C:/FW/edk2-ws/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: C:/FW/edk2-ws/edk2/MyWizardDriver/ComponentName.c

Lab 3: Build and Test Driver

At the VS Command Prompt

```
$> Build
$> RunEmulator.bat
```

Load the UEFI Driver from the shell

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

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

Type: `FS0:\> drivers`

Observe the change in the string that the driver returned

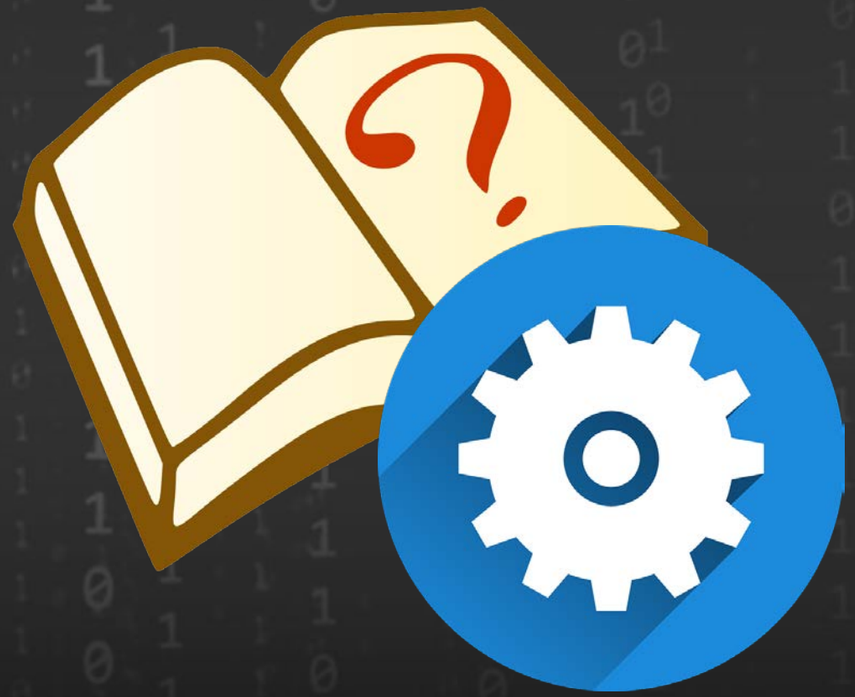
```
92 00000011 ? - -  -  - Usb Mass Storage Driver          UsbMassStorage
93 00000010 B - -  1  1 QEMU Video Driver                QemuVideoDxe
94 00000010 ? - -  -  - Virtio GPU Driver                VirtioGpuDxe
A9 00000000 ? - -  -  - UEFI Sample Driver               \MyWizardDrive
FS0:\> _
```

Exit type `FS0:/ > Reset`

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`

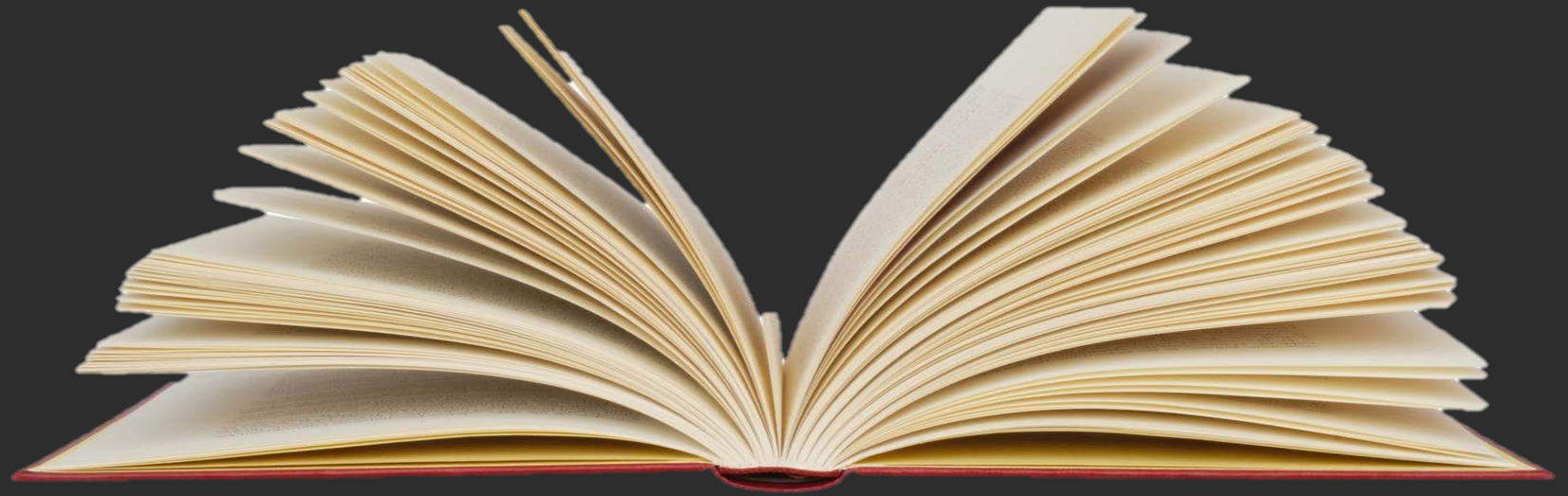
Supported Goals:

- 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** `C:/FW/edk2-ws/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** `C:/FW/edk2-ws/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;
```

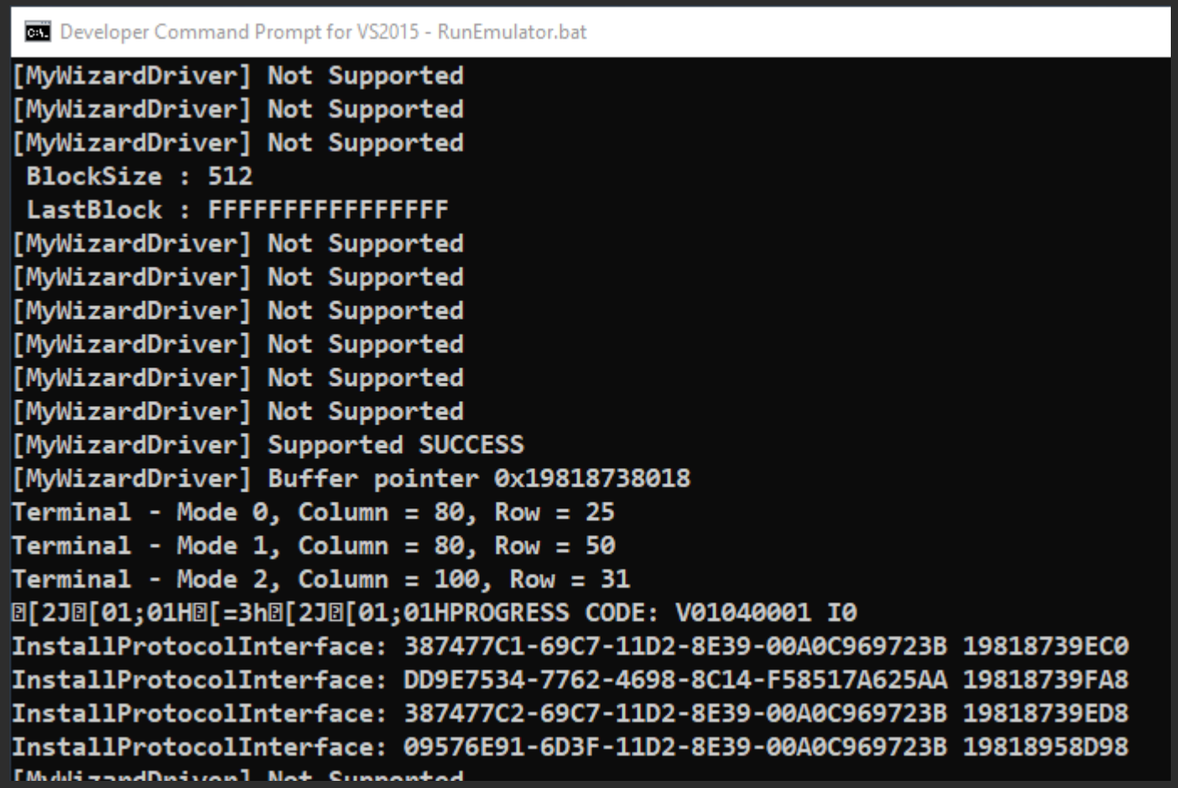
- Notice the Library calls to `AllocateZeroPool()` and `SetMem16()`
- The `Start()` function is where there would be calls to "`gBS-InstallMultipleProtocolInterfaces()`"

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



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%p\r\n", DummyBufferfromStart));
return EFI_SUCCESS;
```

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

Save `C:/FW/edk2-ws/edk2/MyWizardDriver/MyWizardDriver.c`

Lab 4: Build and Test Driver

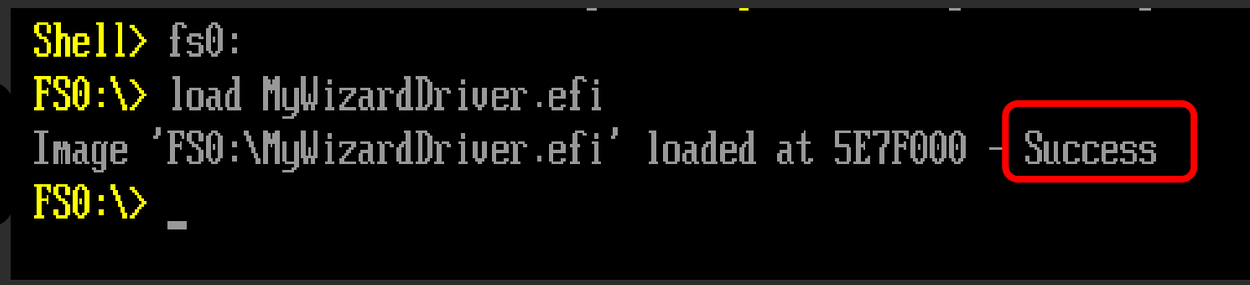
At the VS Command Prompt

```
$> Build
$> RunEmulator.bat
```

Load the UEFI Driver from the shell

At the Shell prompt, type

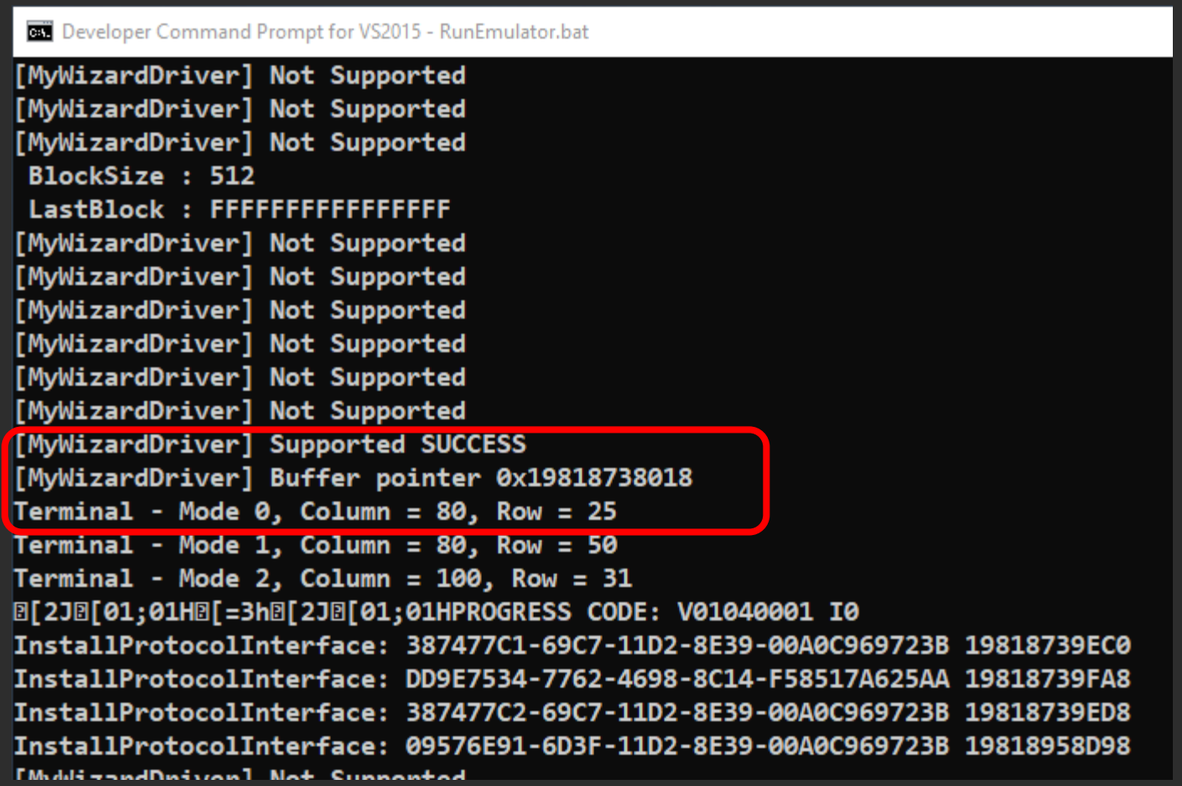
```
Shell> fs0:
FS0:\> load MyWizardDriver.efi
```



Lab 4: Build and Test Driver

- Check the VS 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 type `FS0:/ > Reset`

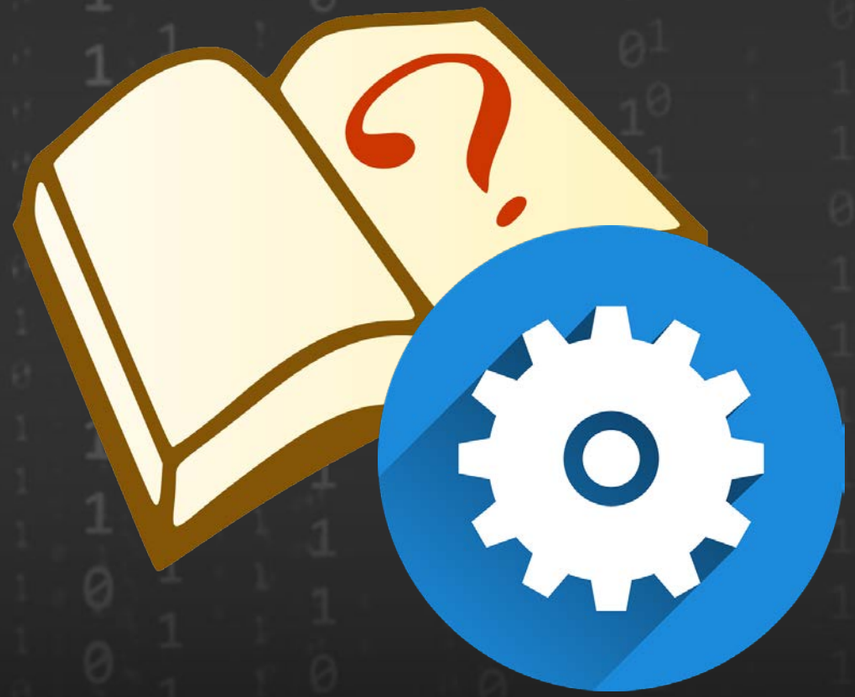


Note: use the right-side scroll bar with mouse to scroll back to see the “Supported SUCCESS”

LAB 5: CREATE A NVRAM VARIABLE

In this lab you'll create a non-volatile UEFI variable (NVRAM), and set and get the variable in the Start 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. In the `Start()` make a call to a new function to set/get the new NVRam Variable
4. 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 “C:/FW/edk2-ws/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 "MyWizardDriverDriverBindingStart ()" function

Copy & Paste at the beginning of the start function to declare a local variable

```
    EFI_STATUS                  Status;  // Declare a local variable Status
```

Copy & Paste the 6 lines: 1) new call to "CreateNVVariable();" , 2-6) if statement with DEBUG just before the line "return EFI_SUPPORTED" as below:

```
Status = CreateNVVariable();
if (EFI_ERROR(Status)) {
        DEBUG((EFI_D_ERROR, "[MyWizardDriver] NV Variable already created \r\n"));
    }
    else {
        DEBUG((EFI_D_ERROR, "[MyWizardDriver] Created NV Variable in the Start \r\n"));
    }

return EFI_SUCCESS;
```

Lab 5: Update MyWizardDriver.c

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

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

    BufferSize = sizeof (MYWIZARDDRIVER_CONFIGURATION);
    Status = gRT->GetVariable(
        mVariableName,
        &mMyWizardDriverVarGuid,
        NULL,
        &BufferSize,
        mMyWizDrv_Conf
        );
    if (EFI_ERROR(Status)) {  // Not definded 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;
}
```

- Note: the `gRT->GetVariable` and `gRT->SetVariable` use Runtime services table
- The Runtime Services Table was not automatically included with the Driver Wizard

Lab 5: Update MyWizardDriver.h

Open "C:/FW/edk2-ws/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 "C:/FW/edk2-ws/edk2/ MyWizardDriver/MyWizardDriver.h"

Save "C:/FW/edk2-ws/edk2/ MyWizardDriver/MyWizardDriver.c"

Lab 5: Build and Test Driver

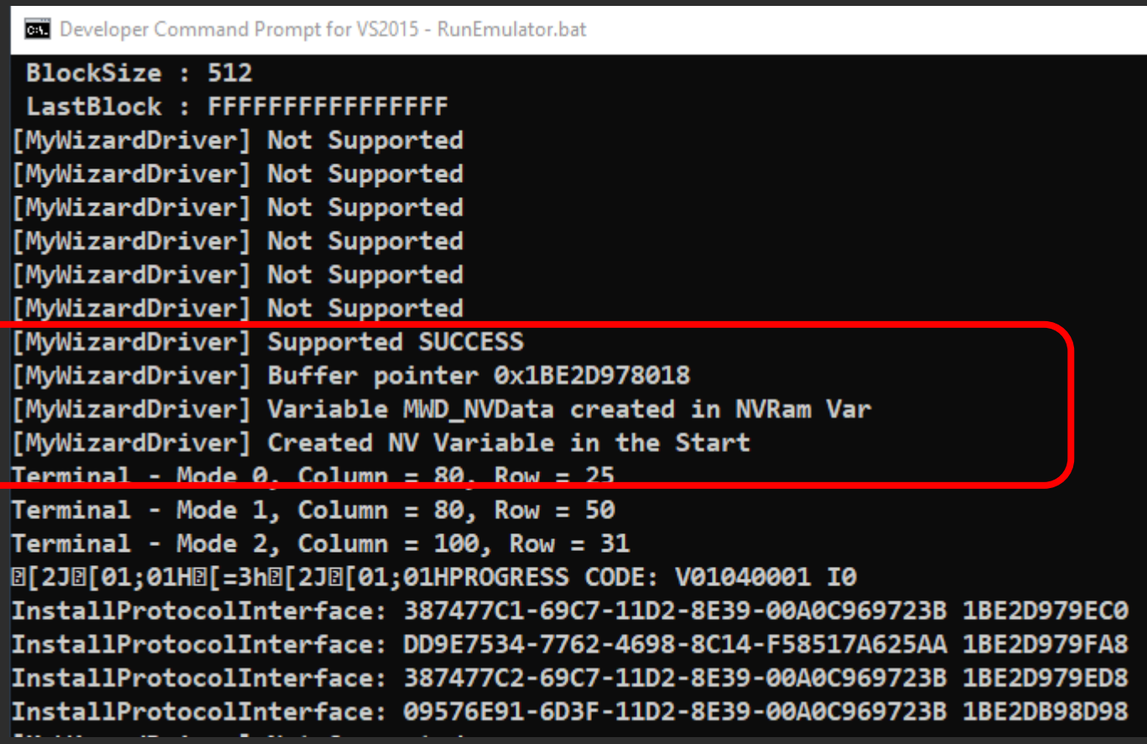
At the VS Command Prompt

```
$> Build
$> RunEmulator.bat
```

Load the UEFI Driver

```
Shell> fs0:
FS0:\> load MyWizardDriver.efi
```

Observe the Buffer address returned by the debug statement in the VS Command window and the new NV Variable was created



Note: use the right-side scroll bar with mouse to scroll back to see the “Supported SUCCESS”

Lab 5: Verify Driver

Use the Buffer address pointer in the previous slide then use the “mem” command

At the Shell prompt, type `FS0:\> mem 0x1be2d978018`

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

```
FS0:\> mem 1be2d978018
Memory Address 000001BE2D978018 200 Bytes
  2D978018: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00  *B.B.B.B.B.B.B.B.*
  2D978028: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00  *B.B.B.B.B.B.B.B.*
  2D978038: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00  *B.B.B.B.B.B.B.B.*
  2D978048: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00  *B.B.B.B.B.B.B.B.*
  2D978058: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00  *B.B.B.B.B.B.B.B.*
  2D978068: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00  *B.B.B.B.B.B.B.B.*
  2D978078: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00  *B.B.B.B.B.B.B.B.*
  2D978088: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00  *B.B.B.B.B.B.B.B.*
  2D978098: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00  *B.B.B.B.B.B.B.B.*
  2D9780A8: 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 new the NVRAM variable "MWD_NVData" was created and filled with 0x00s

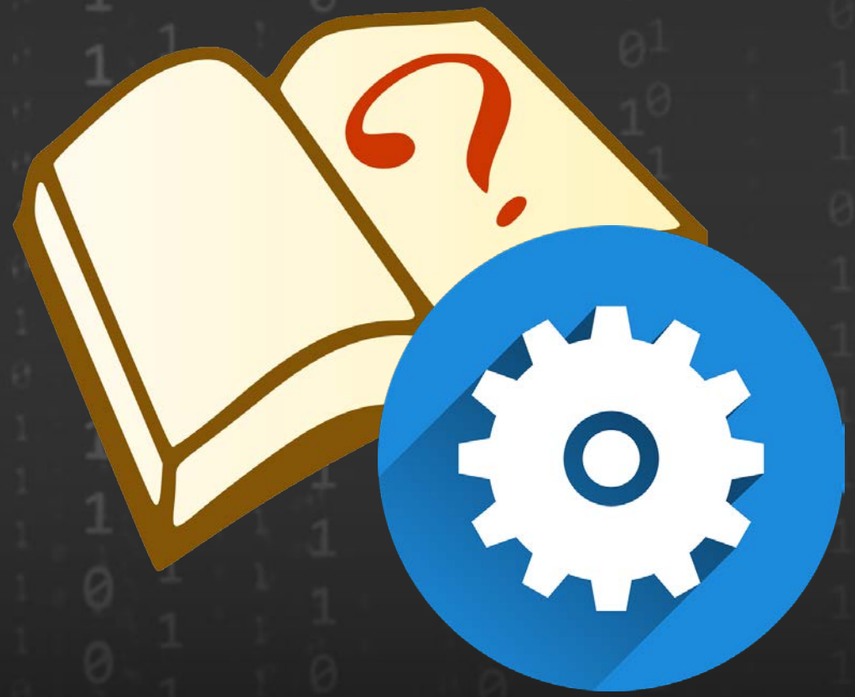
```
FS0:\> dmpstore -all -b
Variable NV+BS '363729F9-35FC-40A6-AFC8-E8F54911F1D6:MWD_NVData' DataSize = 0x2B

  00000000: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  *................*
  00000010: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  *................*
  00000020: 00 00 00 00 00 00 00 00-00 00 00                 *...........*
```

Exit type `FS0:/ > Reset`

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 "C:/FW/edk2-ws/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

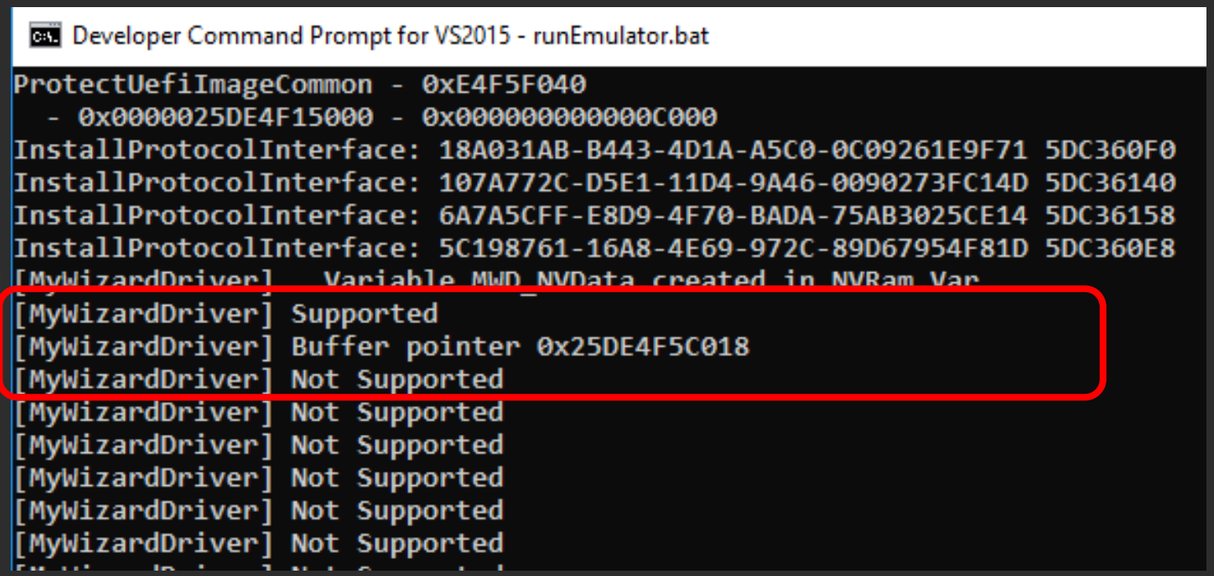
At the VS Command Prompt

```
$> Build
$> RunEmulator.bat
```

Load the UEFI Driver

```
Shell> fs0:
FS0:\> load MyWizardDriver.efi
```

Observe the Buffer address is at `0x25DE4F5C018` as this slide example



Lab 6: Verify Driver

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

```
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:\> _
```

Observe the handle is "A9" as this slide example

Type: `mem 0x25DE4F5C018`

```
FS0:\> mem 25de4f5c018
Memory Address 0000025DE4F5C018 200 Bytes
  E4F5C018: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00  *B.
  E4F5C028: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00  *B.
  E4F5C038: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00  *B.
  E4F5C048: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00  *B.
  E4F5C058: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00  *B.
  E4F5C068: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00  *B.
```

Observe the buffer was filled with the "0x0042"

Lab 6: Verify Unload

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

Observe the DEBUG messages from the Unload in the VS Command Window

Type `Drivers` again to verify

```
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 0x25DE4F5C018`

Observe the buffer is now NOT filled

```
FS0:\> Mem 25DE4F5C018
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 Type `FS0:\> reset`

tianocore

Additional Porting

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

tianocore

Summary

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

tianocore

Questions?

Return to Main Training Page

Return to Training Table of contents for next presentation link

tianocore

ACKNOWLEDGEMENTS

Redistribution and use in source (original document form) and 'compiled‘ forms (converted to PDF, epub, HTML and other formats) with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code (original document form) must retain the above copyright notice, this list of conditions and the following disclaimer as the first lines of this file unmodified.

Redistributions in compiled form (transformed to other DTDs, converted to PDF, epub, HTML and other formats) must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS DOCUMENTATION IS PROVIDED BY TIANOCORE PROJECT "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL TIANOCORE PROJECT BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS DOCUMENTATION, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright (c) 2021, Intel Corporation. All rights reserved.