*tianocore

UEFI & EDK II TRAINING

How to Write a UEFI Driver - Porting Lab – Windows & Simics

tianocore.org

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

Lesson Objective

tianocore

First Setup for Building EDK II, See Lab Setup then
Platform Build Lab for Simics

- Compile a UEFI driver template created from UEFI Driver Wizard

- Test driver w/ Simics QSP Board 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

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
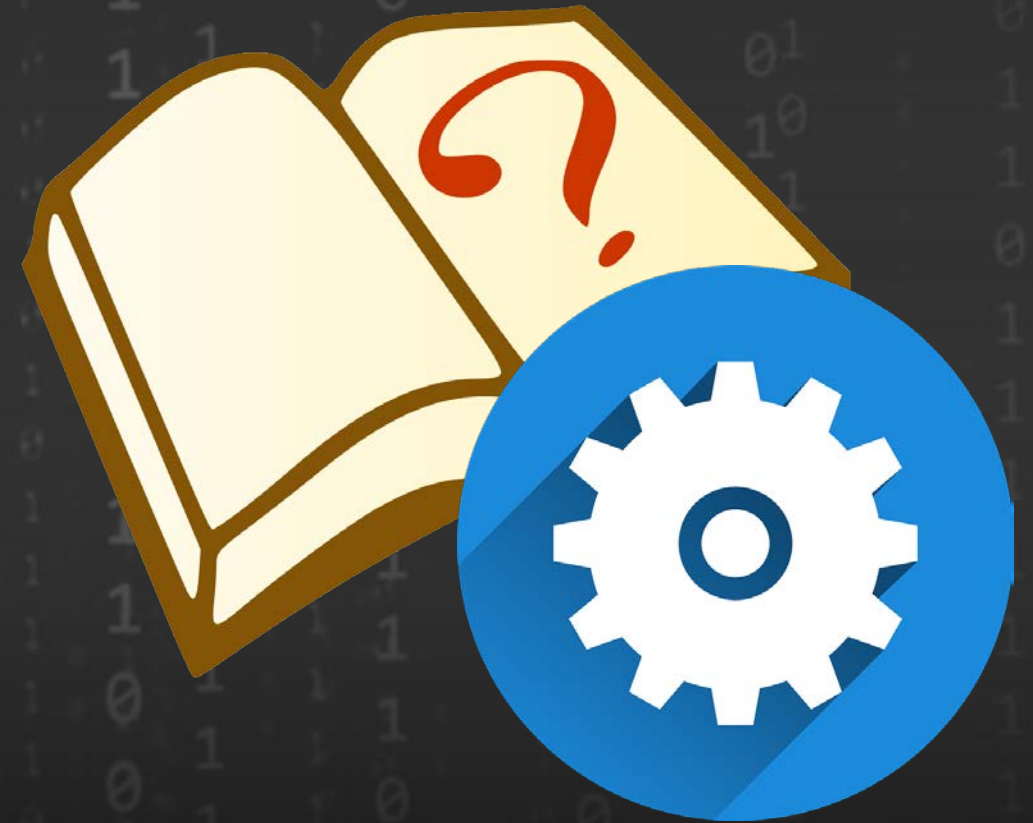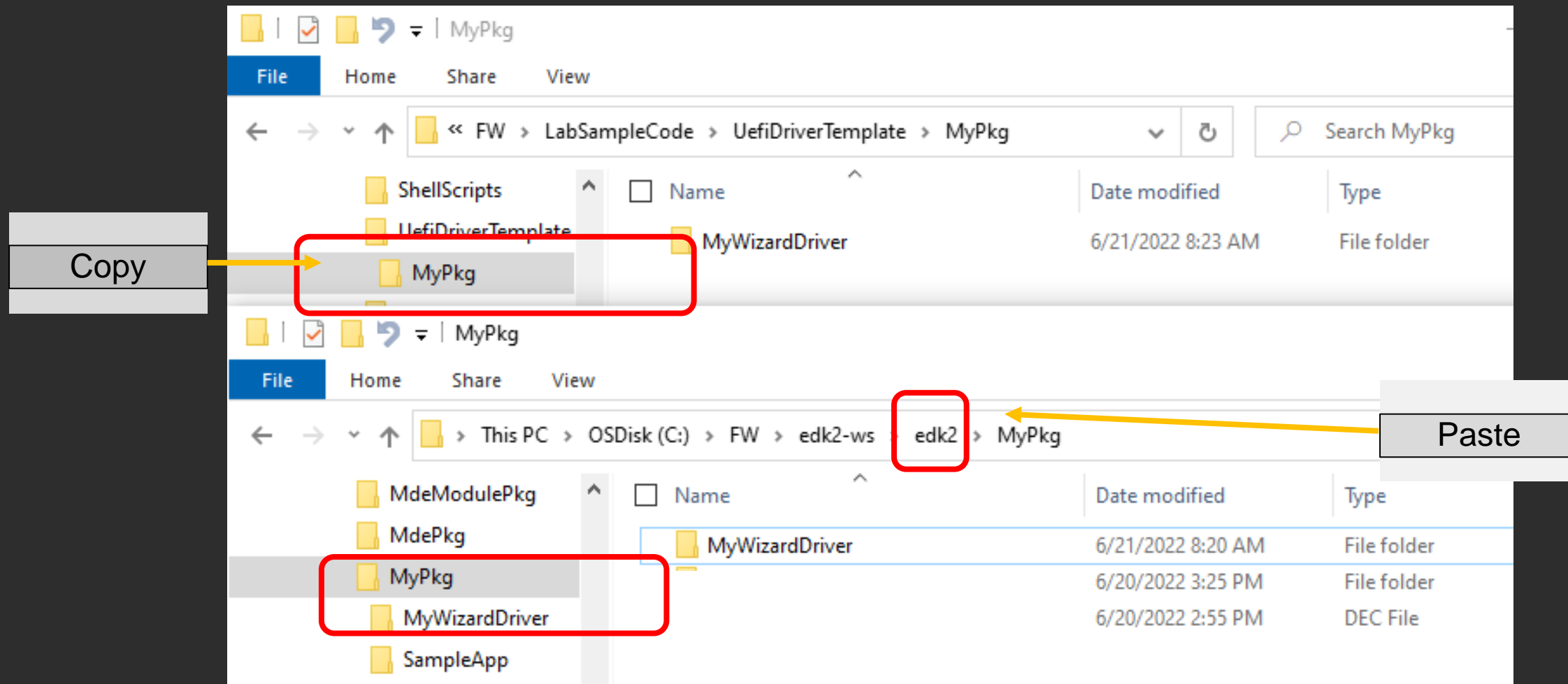
Lab 1: Get UEFI Driver Template

If UEFI Driver Wizard does not work:
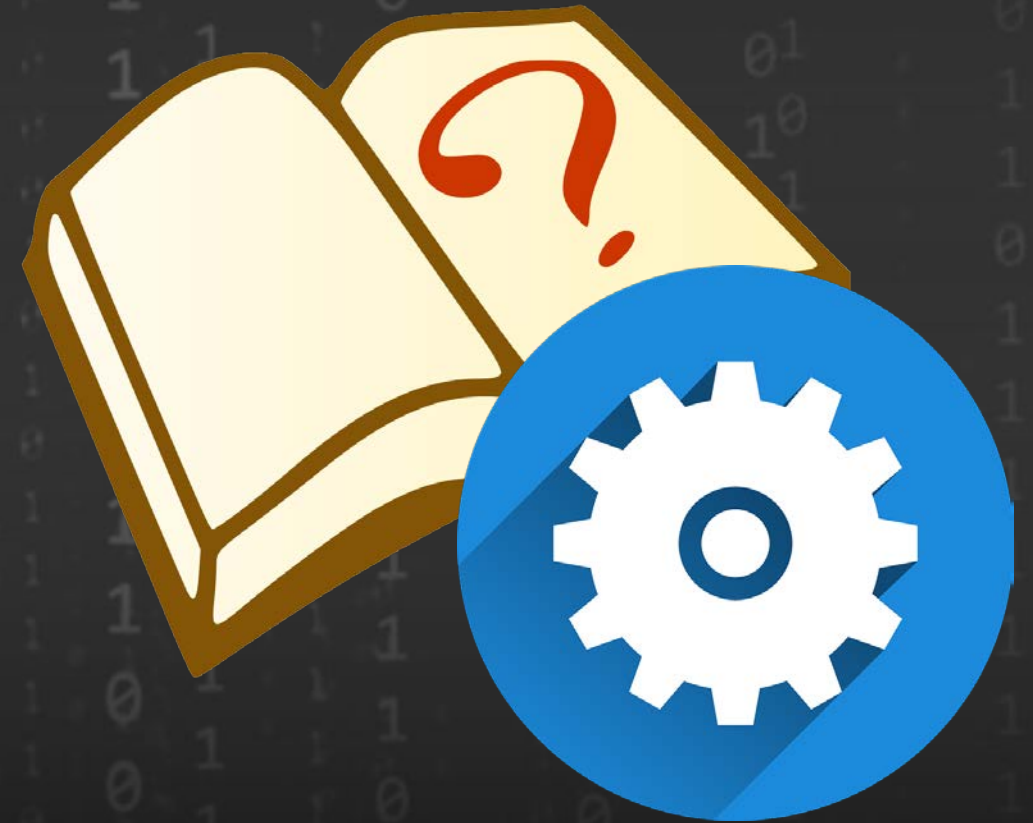1. **Copy** the directory MyPkg from
    `. . .\FW\LabSampleCode\UefiDriverTemplate to  C:\FW\edk2-ws\edk2`

Overwrite If SampleApp Lab Completed

Review <u>UEFI Driver Wizard Lab</u> 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

tianocore

Perform Lab Setup and then Platform Build Lab for Simics from previous Labs

- **Open**
`edk2-platforms/Platform/Intel/SimicsOpenBoardPkg/BoardX58Ich10/OpenBoardPkg.dsc`

- **Add** the following to the [Components] section:
  - Hint: add to the last module in the [Components] section

```
# Add new modules here
  MyPkg/MyWizardDriver/MyWizardDriver.inf
```

- **Save** and close the file `OpenBoardPkg.dsc`

Lab 2: Build the UEFI Driver

- Open the Visual Studio command prompt
- Build the Simics BoardX58Ich10

```
$> cd C:\fw\edk2-ws\edk2-platforms\Platform\Intel
$> python build_bios.py -p BoardX58Ich10  -t VS20XX
```

Where *XX* is 15x86 or 17 or 19

Copy
C:\fw\edk2-ws\Build\SimicsOpenBoardPkg\BoardX58Ich10\DEBUG_VS20XX\FV\BOARDX58ICH10.fd
To
%USERPROFILE%\AppData\Local\Programs\Simics\simics-qsp-x86-6.0.57\targets\qsp-x86\images

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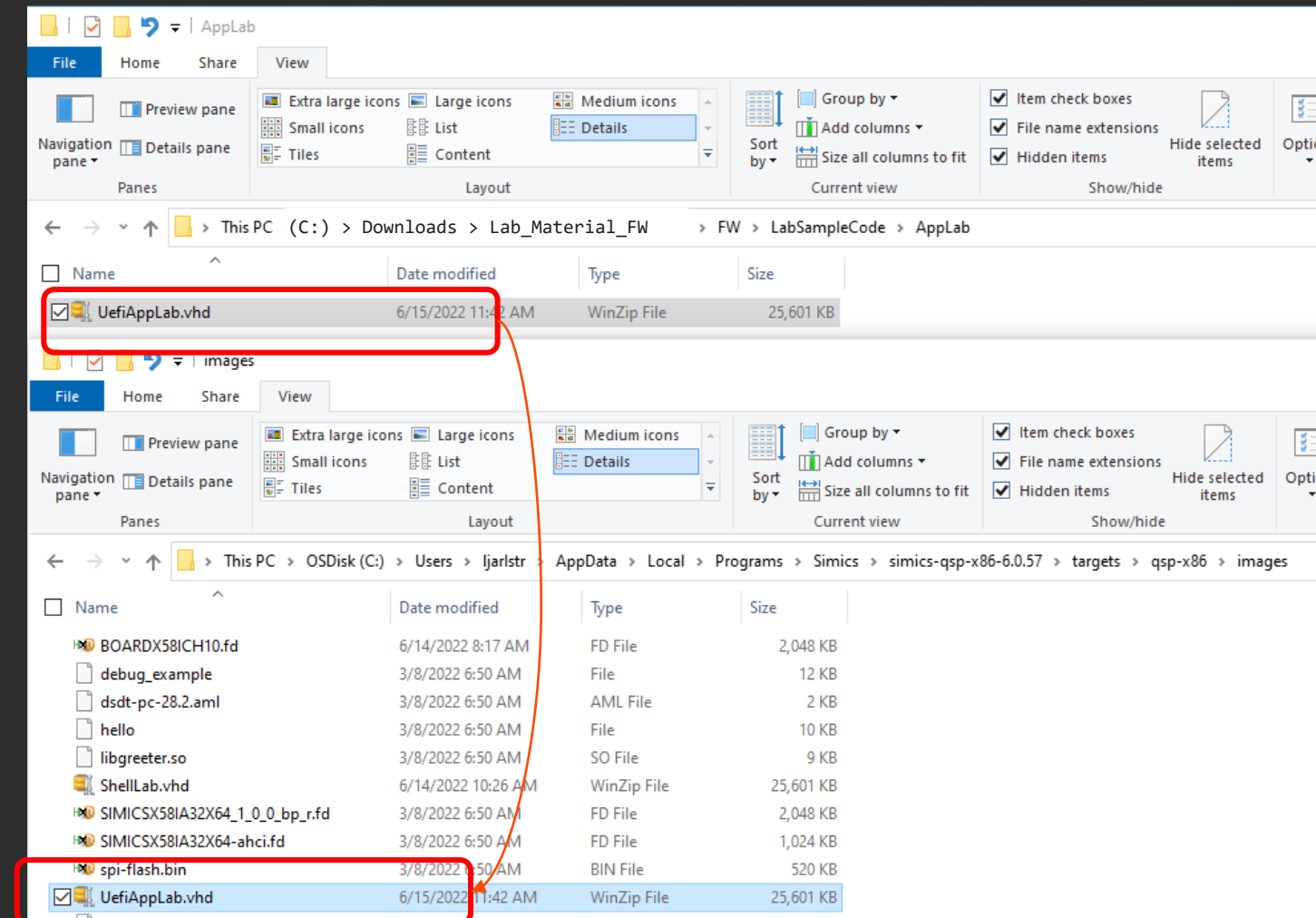
Copy UefiAppLab.vhd file

Copy the UefiAppLab.vhd

From:
.../Lab_Material_FW/FW/LabSampleCode/
AppLab/UefiAppLab.vhd

to

%USERPROFILE%\AppData\Local\Pr
ograms\Simics\simics-qsp-x86-
6.0.57\targets\qsp-x86\images

Update the Simics Script to Use the UefiAppLab.vhd image as a file system

Edit the file: qsp-modern-core.simics from

%USERPROFILE%\
\AppData\Local\Programs\Simics\simics-qsp-cpu-6.0.4\targets\qsp-x86\qsp-modern-core.simics

Add the following Line:

```
$disk1_image="%simics%/targets/qsp
-x86/images/UefiAppLab.vhd"
```

Before the "`run-command-file`" line

Save qsp-modern-core.simics

File: qsp-modern-core.simics

```
Decl{
decl {
! Script that runs the Quick Start Platform (QSP) with a modern
!   processor core.

params from "%simics%/targets/qsp-x86/qsp-clear-linux.simics"
default cpu_comp_class = "x86QSP2"
default num_cores = 2
default num_threads = 2
}
$disk1_image="%simics%/targets/qsp-x86/images/UefiAppLab.vhd"

run-command-file "%simics%/targets/qsp-x86/qsp-clear-linux.simics"
```

Update UefiAppLab.vhd File

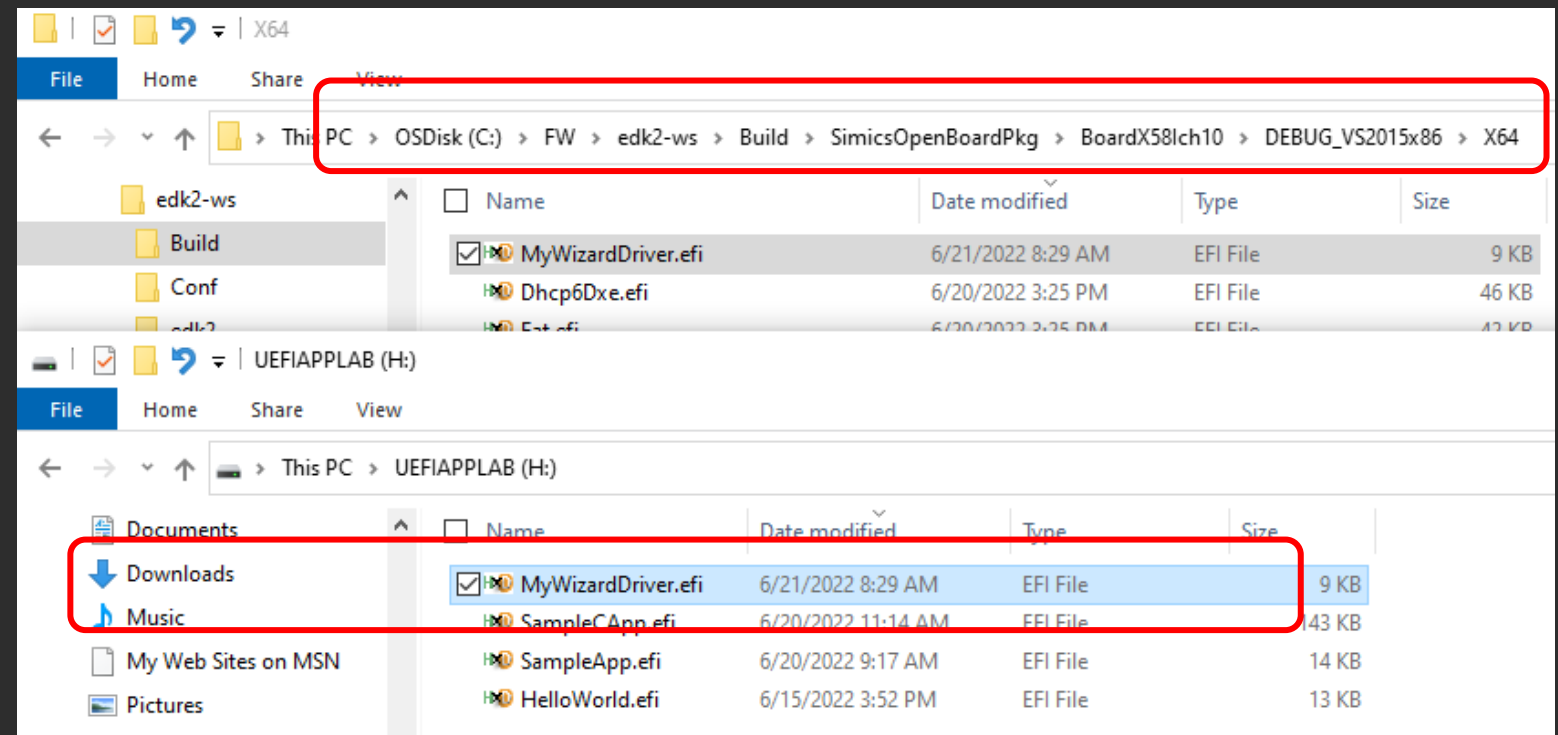Mount the UefiAppLab.vhd using Disk Manager: How To Mount VHD Link

Copy `MyWizardDriver.efi`
`C:\FW\edk2-ws\Build\SimicsOpenBoardPkg\BoardX58Ich10\DEBUG_VS20`*`XX`*`\X64\MyWizardDriver.efi`

Where *XX* is 15x86 or 17 or 19

To
X:\UEFIAPPLAB\ (where *X* is the VHD Drive)

Detach UefiAppLab.vhd
 (can keep open for other Labs)

Run the `qsp-modern-core` script from Windows Command Prompt :

```
$> .\simics targets/qsp-x86/qsp-modern-core.simics
simics> run
```

Press "F2" at the logo, then Select "Boot Manger" followed by "EFI Internal Shell"

At the UEFI Shell prompt

```
Shell> Fs1:
FS1:\> Load MyWizardDriver.efi
```

```
Shell> FS1:
FS1:\> load MyWizardDriver.efi
Image 'FS1:\MyWizardDriver.efi' loaded at DDD3B000 - Success
FS1:\> _
```

tianocore

At the shell prompt Type: `FS1:\> drivers`

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

```
96 0000000A D - - 1  - PS/2 Keyboard Driver        Ps2KeyboardDxe
97 00000010 B - - 1  1 QEMU Video Driver           QemuVideoDxe
98 00002501 B X X 1  1 Intel(R) Gigabit 0.0.25.1   UndiDxe
FF 0000000A ? - - -  - MyWizardDriver              \MyWizardDriver.efi
FS1:\> _
```

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

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

```
FS1:\> dh -d ff
FF: SupportedEfiSpecVersion(0x00020046) ComponentName2 ComponentName DriverBinding HiiPackageList Im
ageDevicePath(..,0xB800)/\MyWizardDriver.efi) LoadedImage(\MyWizardDriver.efi)
    Driver Name [FF]    : MyWizardDriver
    Driver Image Name   : \MyWizardDriver.efi
    Driver Version      : 0000000A
    Driver Type         : <Unknown>
    Configuration       : NO
    Diagnostics         : NO
    Managing            : None
FS1:\> _
```

tianocore

At the shell prompt using the handle from the drivers command,
Type: `FS1:/ > unload ff`
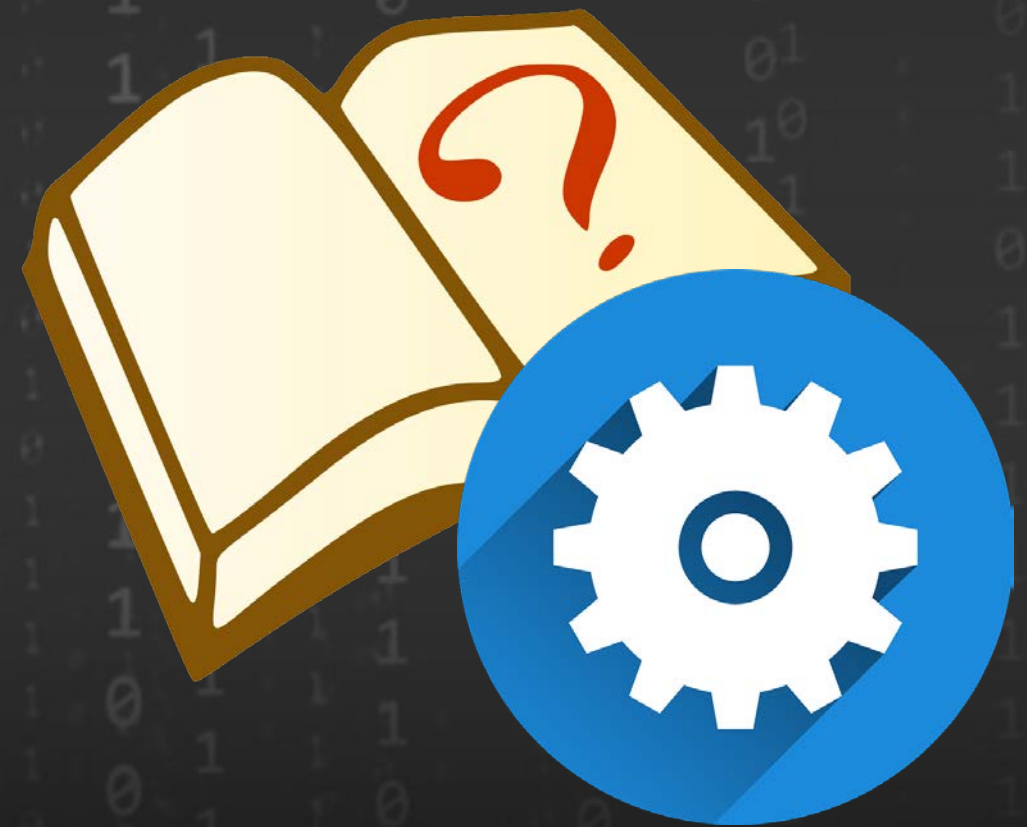
See example screenshot
Type: `drivers` again

Notice results of unload command

```
FS1:\> unload ff
Unload - Handle [DDFC8418].   [y/n]?
y
Unload - Handle [DDFC8418] Result Success.
FS1:\> _
```

Exit Simics `simics> stop, simics> quit`

LAB 3: COMPONENT NAME

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

tianocore

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

Lab 3: Build and Test Driver

1. At the VS Command Prompt, Re-Build BoardX58Ich10

```
$> Cd C:\FW\edk2-ws\edk2-platforms\Platform\Intel\
$> python build_bios.py -p BoardX58Ich10  -t VS20XX
```

2. Copy MyWizardDriver.efi from the build directory to the VHD Disk

Copy ..\Build\SimicsOpenBoardPkg\BoardX58Ich10\DEBUG_VS20XX\X64\MyWizardDriver.efi UefiAppLab

3. Run the `qsp-modern-core` script from Windows Command Prompt :

```
$> .\simics targets/qsp-x86/qsp-modern-core.simics
simics> run
```

4. At the Shell, Load Driver

```
Shell> fs1:
FS1:\> load MyWizardDriver.efi
```

```
96 0000000A D - - 1  - PS/2 Keyboard Driver          Ps2KeyboardDxe
97 00000010 B - - 1  1 QEMU Video Driver             QemuVideoDxe
98 00002501 B X X 1  1 Intel(R) Gigabit 0.0.25.1     UndiDxe
FF 0000000A ? - - -  - UEFI Sample Driver            \MyWizardDriver.efi
FS1:\> _
```
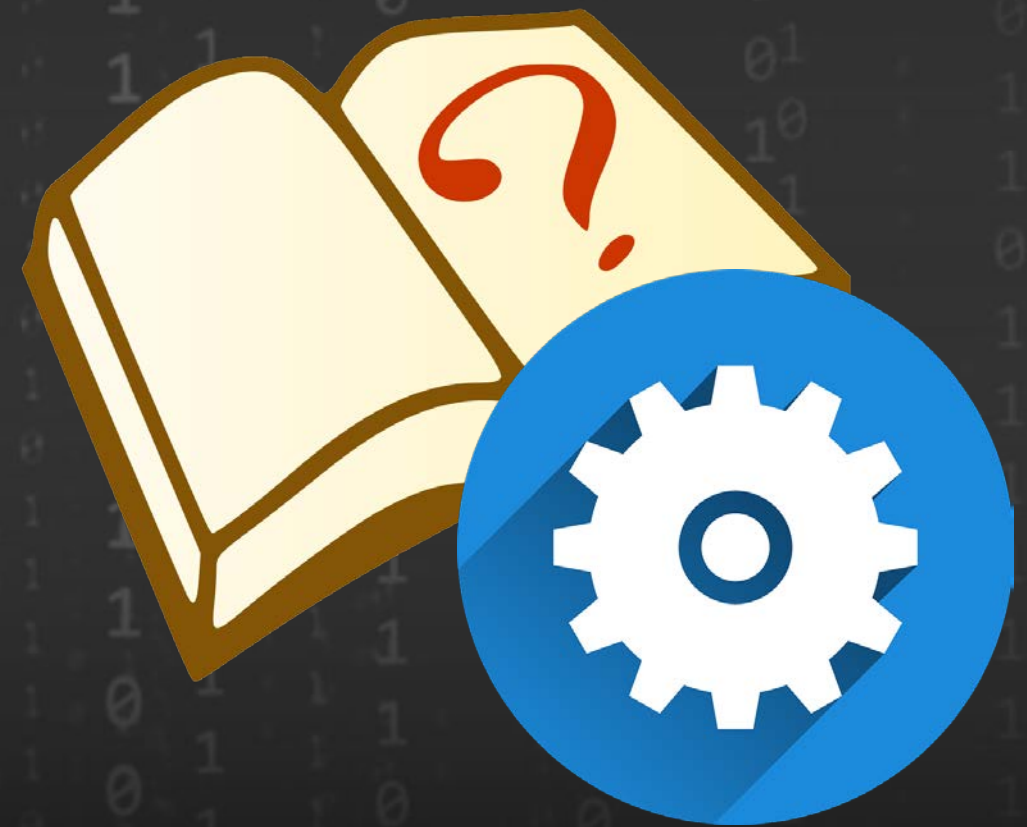
5. Type Drivers `FS1:\> Drivers`

6. Exit Simics `simics> stop`, `simics> quit`   Notice the Name has changed in the list of drivers

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

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

www.tianocore.org

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

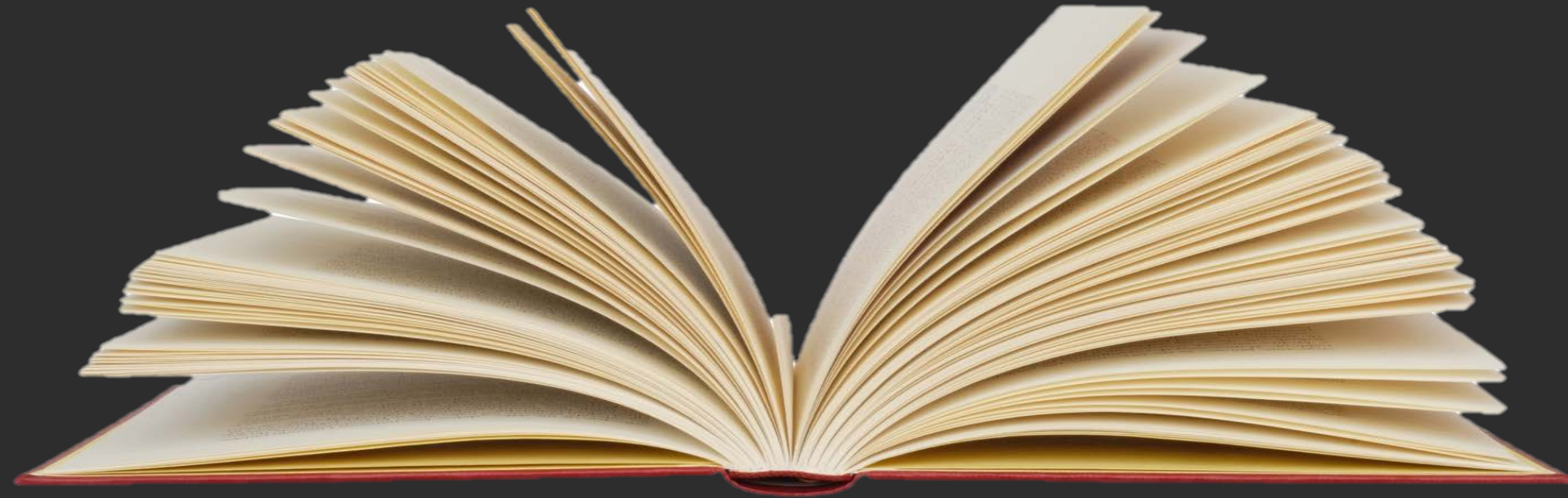- Helps locate a protocol's specific GUID through UEFI Boot Services' function

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/MyPkg/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 paste (next slide)

This code checks for a specific protocol before returning a status for the supported function (EFI_SUCCESS if the protocol GUID exists).

Copy & Paste the following code for the supported function

`MyWizardDriverDriverBindingSupported()`:

```
EFI_STATUS                        Status;
EFI_PCI_IO_PROTOCOL        *UsbIo;
Status = gBS->OpenProtocol (
                ControllerHandle,
                &gEfiUsbIoProtocolGuid,
                (VOID **)&UsbIo,
                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,
      &gEfiUsbIoProtocolGuid,
      This->DriverBindingHandle,
      ControllerHandle
      );

   return Status;
```

Lab 4: Notice UEFI Driver Wizard Includes

- **Open** `C:/FW/edk2-ws/edk2/MyPkg/MyWizardDriver/MyWizardDriver.h`
- **Notice** the following include statement is already added by the driver wizard:

```
// Consumed Protocols
//
#include <Protocol/UsbIo.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;
}
```

Copy & Paste   the following code for the start function

`MyWizardDriverDriverBindingStart():`

```
BOOLEAN FirstAlloc = FALSE;

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

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

if (FirstAlloc) {
    SetMem16 (DummyBufferfromStart, (DUMMY_SIZE * sizeof(CHAR16)), 0x004A);  // 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


```
and cause SInstallProtocolInterface: 387477C2-69C7-11D2-8E39-00A0C969723B DE101
320
InstallProtocolInterface: 752F3136-4E16-4FDC-A22A-E5F46812F4CA DDFD0F18
InstallProtocolInterface: 6302D008-7F9B-4F30-87AC-60C9FEF5DA4E DD1368C0
FSOpen: Open '\' Success
FSOpen: Open '.' Success
FSOpen: Open '.' Success
FSOpen: Open '\MyWizardDriver.efi' Success
FSOpen: Open '\MyWizardDriver.efi' Success
[Security] 3rd party image[0] can be loaded after EndOfDxe: PciRoot(0x0)/Pci(0x1
F,0x2)/Sata(0x2,0xFFFF,0x0)/HD(1,GPT,23D35AB0-9FBF-44CB-8EAB-DDBE10857B20,0x80,0
xB800)/\MyWizardDriver.efi.
InstallProtocolInterface: 5B1B31A1-9562-11D2-8E3F-00A0C969723B DDF45240
Loading driver at 0x000DDD3B000 EntryPoint=0x000DDD3B4AC MyWizardDriver.efi
InstallProtocolInterface: BC62157E-3E33-4FEC-9920-2D3B36D750DF DDF46018
InstallProtocolInterface: 6A1EE763-D47A-43B4-AABE-EF1DE2AB56FC DDD3D260
ProtectUefiImageCommon - 0xDDF45240
  - 0x00000000DDD3B000 - 0x00000000000023A0
InstallProtocolInterface: 18A031AB-B443-4D1A-A5C0-0C09261E9F71 DDD3CF28
InstallProtocolInterface: 107A772C-D5E1-11D4-9A46-0090273FC14D DDD3CF58
InstallProtocolInterface: 6A7A5CFF-E8D9-4F70-BADA-75AB3025CE14 DDD3CF70
ExtractConfig: BlockToConfig(): Invalid Parameter, Progress="<null string>"
InstallProtocolInterface: 5C198761-16A8-4E69-972C-89D67954F81D DDD3CF20
PROGRESS CODE: V02010000 I0
```

Simics Serial Console Output Debug Messages

Lab 4: Add Debug Statements Supported()

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

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

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

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

Lab 4: Add Debug Statements Start()

Copy & Paste the following DEBUG macro for the Start function just after the `SetMem16` function call

```
if (FirstAlloc) {
    SetMem16(DummyBufferfromStart, (DUMMY_SIZE * sizeof(CHAR16)), 0x004A);
    DEBUG((DEBUG_INFO, "\n*******\n***[MyWizardDriver] Buffer 0x%p  ***\n*******\n",
        DummyBufferfromStart));
}
```

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

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

Lab 4: Build and Test Driver

1. At the VS Command Prompt, Re-Build BoardX58Ich10

```
$> Cd C:\FW\edk2-ws\edk2-platforms\Platform\Intel\
$> python build_bios.py -p BoardX58Ich10  -t VS20XX
```

2. Copy MyWizardDriver.efi from the build directory to the VHD Disk

Copy ..\Build\SimicsOpenBoardPkg\BoardX58Ich10\DEBUG_VS20XX\X64\MyWizardDriver.efi UefiAppLab

3. Run the `qsp-modern-core` script from Windows Command Prompt :

```
$> .\simics targets/qsp-x86/qsp-modern-core.simics
simics> run
```

4. At the Shell, Load Driver

```
Shell> fs1:
FS1:\> load MyWizardDriver.efi
```

```
Shell> fs1:
FS1:\> load MyWizardDriver.efi
Image 'FS1:\MyWizardDriver.efi' loaded at DDD3B000 - Success
FS1:\> _
```

Lab 4: Build and Test Driver

- Check the Simics Com[0] 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.
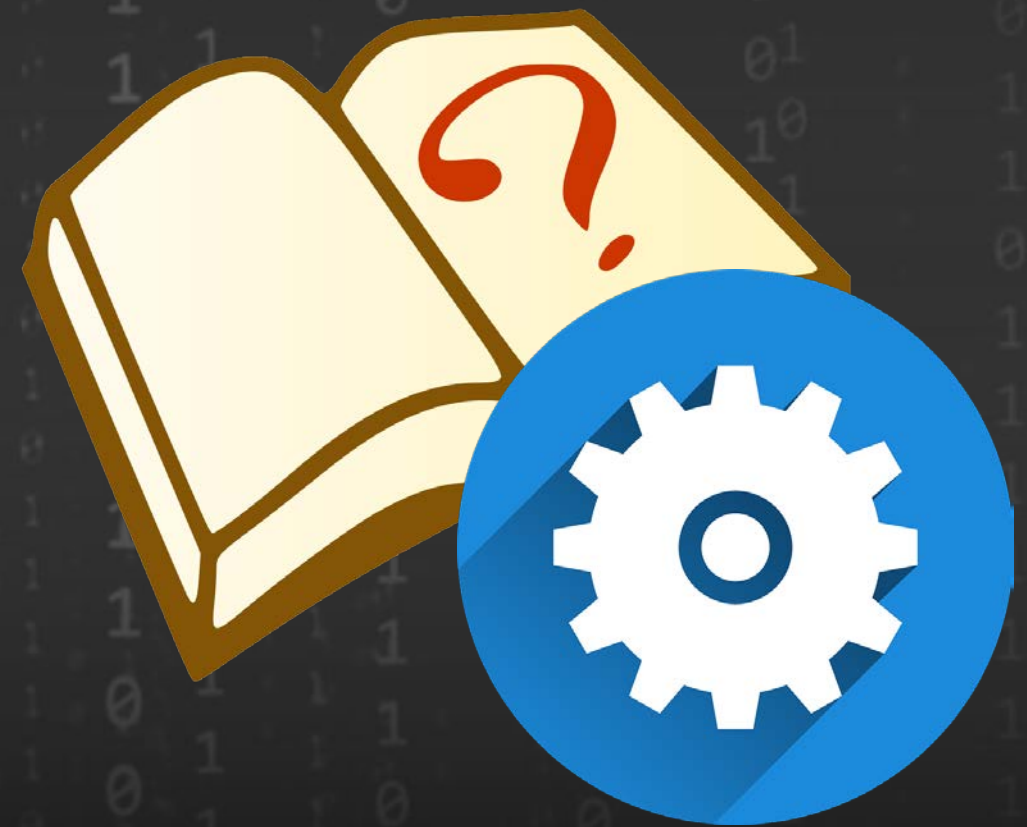
Exit Simics `simics> stop, simics> quit`

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.

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

Copy, Paste and then **Save** this file in the `C:/FW/edk2-ws/edk2/MyPkg/MyWizardDriver` Directory

```c
#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 \
    }
#define MYWIZARDDRIVER_STRING_SIZE    0x1A

#pragma pack(1)
typedef struct {
    UINT16  MyWizardDriverStringData[MYWIZARDDRIVER_STRING_SIZE];
    UINT8   MyWizardDriverHexData;
    UINT8   MyWizardDriverBaseAddress;
    UINT8   MyWizardDriverChooseToEnable;
    CHAR16  *MyWizardDriverNvRamAddress;
  } MYWIZARDDRIVER_CONFIGURATION;

#pragma pack()
#endif
```

Lab 5: Update MyWizardDriver.c

Open "C:/FW/edk2-ws/edk2/MyPkg/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_SUCCESS" and inside the "if (FirstAlloc)" as below:

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

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 init before call
                );
            DEBUG((DEBUG_INFO, "***[MyWizardDriver] Variable %s created in NVRam Var\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/MyPkg/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/MyPkg/MyWizardDriver/MyWizardDriver.h"

Lab 5- Improvements(1) MyWizardDriver.c

In Lab 4 every time the Supported function was called a Debug message was printed to the Serial port resulting in many messages to examine. Instead, use a different Debug message type for the "Not Supported" Debug message

In the `MyWizardDriverDriverBindingSupported` function after the call to "`OpenProtocol`" fails use "`DEBUG_VERBOSE`" instead of "`DEBUG_INFO`" This can be changed by setting the PCD message flag in the DSC file.


```
FSOpen: Open '\MyWizardDriver.efi' Success
FSOpen: Open '\MyWizardDriver.efi' Success
[Security] 3rd party image[0] can be loaded after EndOfDxe: PciRoot(0x
F,0x2)/Sata(0x2,0xFFFF,0x0)/HD(1,GPT,23D35AB0-9FBF-44CB-8EAB-DDBE10857
xB800)/\MyWizardDriver.efi.
InstallProtocolInterface: 5B1B31A1-9562-11D2-8E3F-00A0C969723B DDF453C
Loading driver at 0x000DDD3B000 EntryPoint=0x000DDD3B4CC MyWizardDrive
InstallProtocolInterface: BC62157E-3E33-4FEC-9920-2D3B36D750DF DDF46B9
InstallProtocolInterface: 6A1EE763-D47A-43B4-AABE-EF1DE2AB56FC DDD3D72
ProtectUefiImageCommon - 0xDDF453C0
  - 0x00000000DDD3B000 - 0x0000000000002860
InstallProtocolInterface: 18A031AB-B443-4D1A-A5C0-0C09261E9F71 DDD3D3A
InstallProtocolInterface: 107A772C-D5E1-11D4-9A46-0090273FC14D DDD3D3D
InstallProtocolInterface: 6A7A5CFF-E8D9-4F70-BADA-75AB3025CE14 DDD3D3F
ExtractConfig: BlockToConfig(): Invalid Parameter, Progress="<null str
InstallProtocolInterface: 5C198761-16A8-4E69-972C-89D67954F81D DDD3D3A
[MyWizardDriver] Not Supported
[MyWizardDriver] Not Supported
[MyWizardDriver] Not Supported
[MyWizardDriver] Not Supported
[MyWizardDriver] Not Supported
[MyWizardDriver] Not Supported
[MyWizardDriver] Not Supported
[MyWizardDriver] Not Supported
```

```
Status = gBS->OpenProtocol(  . . .
// . . .
if (EFI_ERROR(Status)) {
    DEBUG((DEBUG_VERBOSE, "[MyWizardDriver] Not Supported  \n" ));
    return Status; // Bail out if OpenProtocol returns an error
}
```

40

It is hard to find the Buffer address in the Debug Message.

Before the call to `CreateNVVariable()` in the `MyWizardDriverDriverBindingStart()` Function add the following:

1. Store the address of the Dummy Buffer in the NVRAM Variable

2. Use "StrCpyS" to store the string: "UEFI-Training-Class-MWD" to the NVRAM Variable String

```
//  store the address and string value in the NvRam Variable  -
//  this Allows DMPSTORE to display our buffer address
  mMyWizDrv_Conf_buffer.MyWizardDriverNvRamAddress = DummyBufferfromStart;
  StrCpyS(mMyWizDrv_Conf_buffer.MyWizardDriverStringData,
        (MYWIZARDDRIVER_STRING_SIZE * sizeof(CHAR16)),
        L"UEFI-Training-Class-MWD");
```

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

Lab 5: Build and Test Driver

1. At the VS Command Prompt, Re-Build BoardX58Ich10

```
$> Cd C:\FW\edk2-ws\edk2-platforms\Platform\Intel\
$> python build_bios.py -p BoardX58Ich10  -t VS20XX
```

2. Copy MyWizardDriver.efi from the build directory to the VHD Disk

```
Copy ..\Build\SimicsOpenBoardPkg\BoardX58Ich10\DEBUG_VS20XX\X64\MyWizardDriver.efi UefiAppLab
```

3. Run the `qsp-modern-core` script from Windows Command Prompt :

```
$> .\simics targets/qsp-x86/qsp-modern-core.simics
simics> run
```

4. At the Shell, Load Driver

```
Shell> fs1:
FS1:\> load MyWizardDriver.efi
```
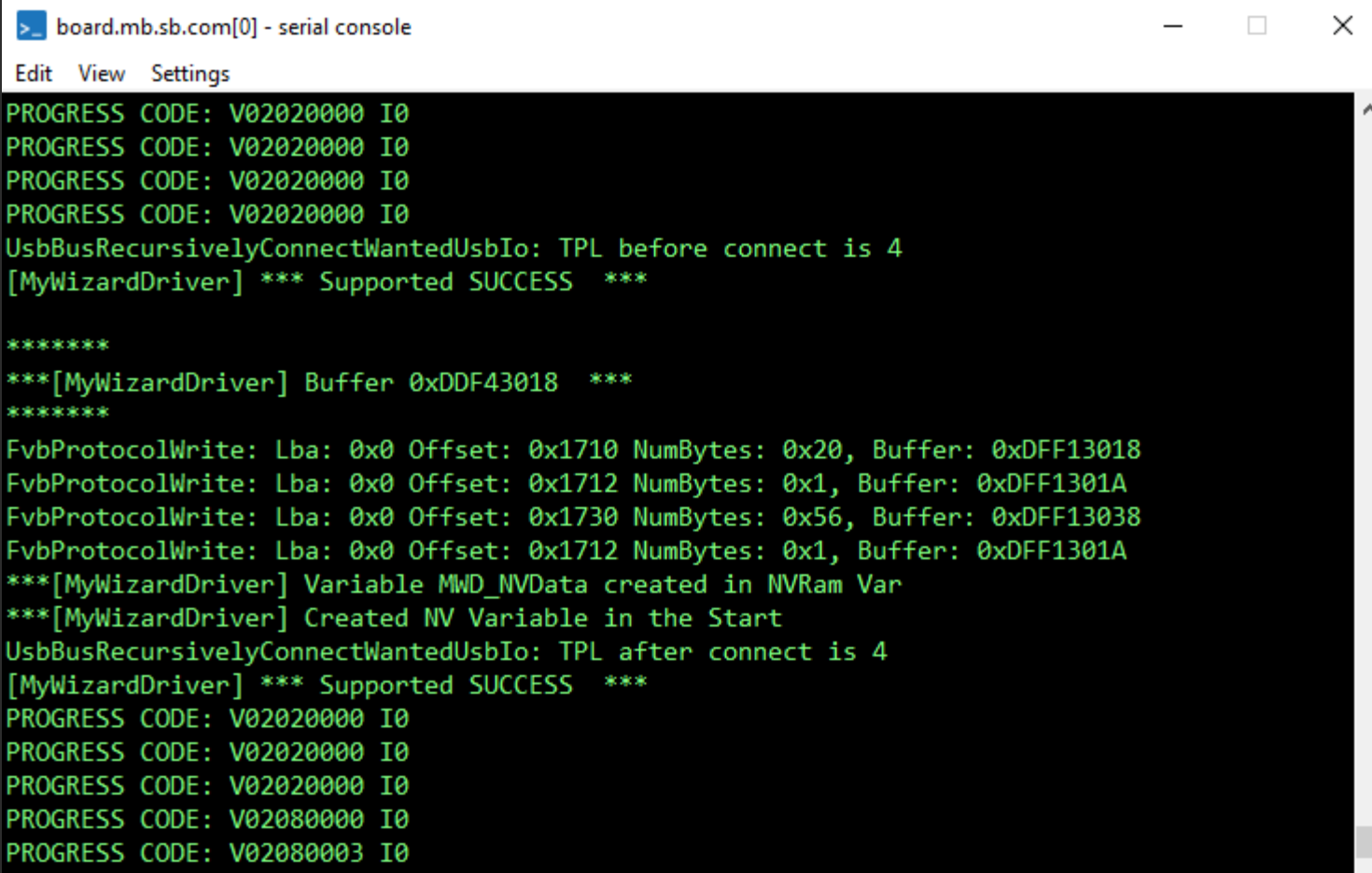
```
Shell> fs1:
FS1:\> load MyWizardDriver.efi
Image 'FS1:\MyWizardDriver.efi' loaded at DDD3B000 - Success
FS1:\> _
```

Observe the Buffer address returned by the debug statement in the Simics Serial Console window and the new NV Variable was created

Also note, the "[MyWizardDriver] Not Supported" Messages are no longer displayed.

To display these, Set the `PcdDebugPrintErrorLevel|0x80400040`
In the DSC file

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

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

At the Shell prompt, type `FS1:\> mem 0ddf43018`

Observe the Buffer is filled with the letter "J" or 0x004A

```
FS1:\> mem 0ddf43018
Memory Address 00000000DDF43018 200 Bytes
  DDF43018: 4A 00 4A 00 4A 00 4A 00-4A 00 4A 00 4A 00 4A 00  *J.J.J.J.J.J.J.J.*
  DDF43028: 4A 00 4A 00 4A 00 4A 00-4A 00 4A 00 4A 00 4A 00  *J.J.J.J.J.J.J.J.*
  DDF43038: 4A 00 4A 00 4A 00 4A 00-4A 00 4A 00 4A 00 4A 00  *J.J.J.J.J.J.J.J.*
  DDF43048: 4A 00 4A 00 4A 00 4A 00-4A 00 4A 00 4A 00 4A 00  *J.J.J.J.J.J.J.J.*
  DDF43058: 4A 00 4A 00 4A 00 4A 00-4A 00 4A 00 4A 00 4A 00  *J.J.J.J.J.J.J.J.*
  DDF43068: 4A 00 4A 00 4A 00 4A 00-4A 00 4A 00 4A 00 4A 00  *J.J.J.J.J.J.J.J.*
  DDF43078: 4A 00 4A 00 4A 00 4A 00-4A 00 4A 00 4A 00 4A 00  *J.J.J.J.J.J.J.J.*
  DDF43088: 4A 00 4A 00 4A 00 4A 00-4A 00 4A 00 4A 00 4A 00  *J.J.J.J.J.J.J.J.*
  DDF43098: 4A 00 4A 00 4A 00 4A 00-4A 00 4A 00 4A 00 4A 00  *J.J.J.J.J.J.J.J.*
  DDF430A8: 4A 00 4A 00 4A 00 4A 00-4A 00 4A 00 4A 00 4A 00  *J.J.J.J.J.J.J.J.*
```

At the Shell prompt, type `FS1:\> dmpstore -all MWD_NVData`

Observe new the NVRAM variable "`MWD_NVData`" was created and filled with the address of the buffer and the string "UEFI-Training-Class-MWD"

```
FS1:\> dmpstore -all MWD_NVData
Variable NV+BS '363729F9-35FC-40A6-AFC8-E8F54911F1D6:MWD_NVData' DataSize = 0x40
  00000000: 55 00 45 00 46 00 49 00-2D 00 54 00 72 00 61 00   *U.E.F.I.-.T.r.a.*
  00000010: 69 00 6E 00 69 00 6E 00-67 00 2D 00 43 00 6C 00   *i.n.i.n.g.-.C.l.*
  00000020: 61 00 73 00 73 00 2D 00-4D 00 57 00 44 00 00 00   *a.s.s.-.M.W.D...*
  00000030: 00 00 00 00 00 00 00 00-18 30 F4 DD 00 00 00 00   *.........0......*
FS1:\> _
```

String

Buffer

Buffer address is: `00 00 DD F4 30 18`

Exit Simics `simics> stop, simics> quit`

Lab 5: More Porting Needed for the Start

At this point the MyWizardDriver does not manage anything.

The next steps would be to install a protocol to manage the Buffer and NVRAM variable.

HandleProtocol(GUID, …)

Handle

Protocol Interface
  Function Pointer
  Function Pointer
  . . .

Handle Database

⭐ **Start function**
Install Protocols to Handle database

UEFI Driver

GUID
Protocol specific functions

GUID
Protocol specific functions
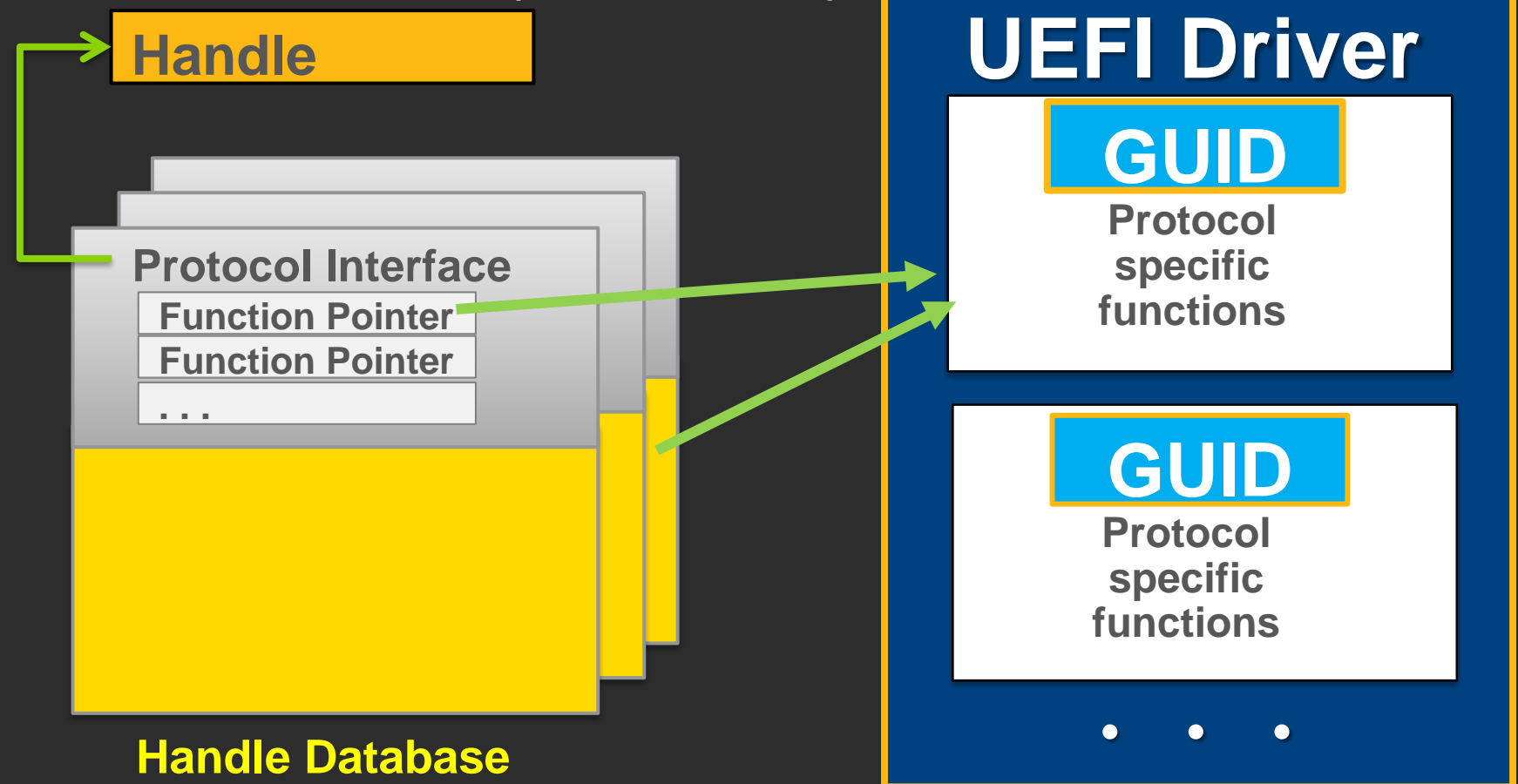
LAB 6: PORT STOP AND UNLOAD

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.
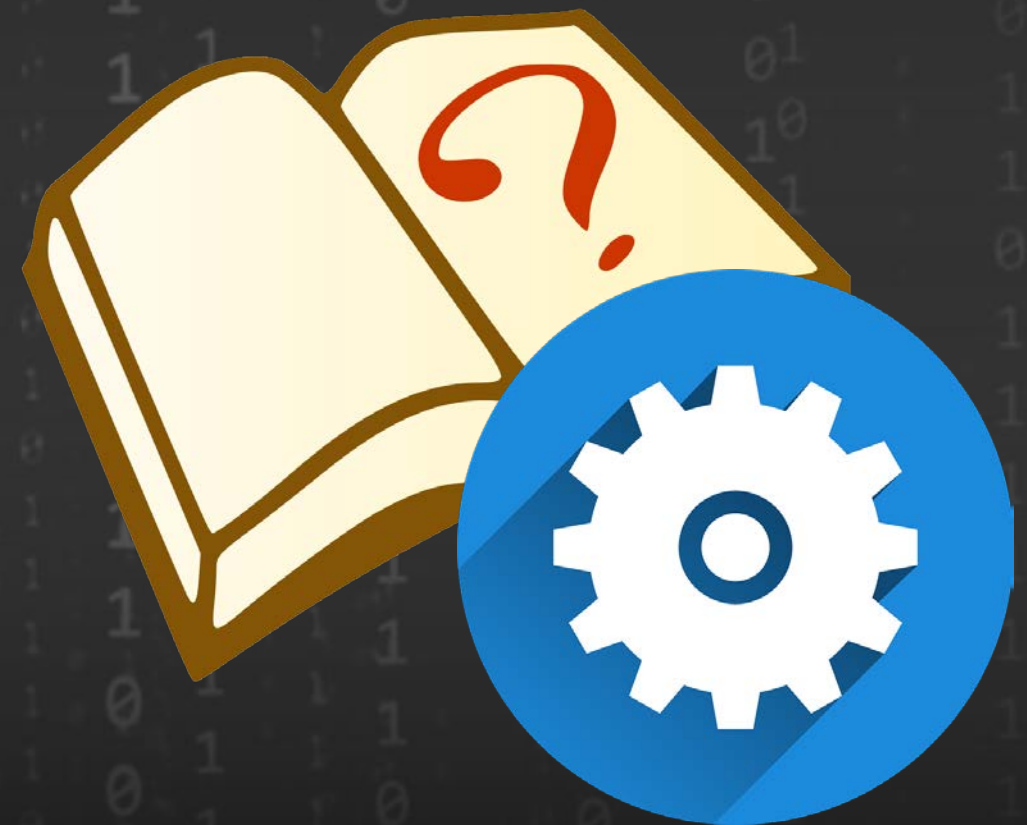
Lab 6: Port the Unload function

Open "C:/FW/edk2-ws/edk2/MyPkg/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\n"));
}
DEBUG((DEBUG_INFO, "[MyWizardDriver] Unload success\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((DEBUG_INFO, "[MyWizardDriver] Stop, clear buffer\n"));
}
DEBUG((DEBUG_INFO, "[MyWizardDriver] Stop, EFI_SUCCESS\n"));

return EFI_SUCCESS;
//  return EFI_UNSUPPORTED;
}
```

Save & Close "MyWizardDriverDriver.c"

tianocore

1. At the VS Command Prompt, Re-Build BoardX58Ich10

```
$> Cd C:\FW\edk2-ws\edk2-platforms\Platform\Intel\
$> python build_bios.py -p BoardX58Ich10  -t VS20XX
```

2. Copy MyWizardDriver.efi from the build directory to the VHD Disk

Copy ..\Build\SimicsOpenBoardPkg\BoardX58Ich10\DEBUG_VS20XX\X64\MyWizardDriver.efi UefiAppLab

3. Run the `qsp-modern-core` script from Windows Command Prompt :

```
$> .\simics targets/qsp-x86/qsp-modern-core.simics
simics> run
```

4. At the Shell, Load Driver

```
Shell> fs1:
FS1:\> load MyWizardDriver.efi
```

Observe the Buffer address is at
0xDDF43018 as this slide example

```
[MyWizardDriver] *** Supported SUCCESS  ***

*******
***[MyWizardDriver] Buffer 0xDDF43018  ***
*******
FvbProtocolWrite: Lba: 0x0 Offset: 0x1710 NumBytes: 0x20, Buffe
FvbProtocolWrite: Lba: 0x0 Offset: 0x1712 NumBytes: 0x1, Buffer
FvbProtocolWrite: Lba: 0x0 Offset: 0x1730 NumBytes: 0x56, Buffe
FvbProtocolWrite: Lba: 0x0 Offset: 0x1712 NumBytes: 0x1, Buffer
***[MyWizardDriver] Variable MWD_NVData created in NVRam Var
***[MyWizardDriver] Created NV Variable in the Start
```

At the Shell prompt, type `FS1:\> drivers`

```
97 00000010 B - -  1  1 QEMU Video Driver
98 00002501 B X X  1  1 Intel(R) Gigabit 0.0.25.1
FF 0000000A ? - -  -  - UEFI Sample Driver
FS1:\> _
```

Observe the handle is "FF" as this slide example

Type: `mem 0xDDF43018`

Observe the buffer was filled with "0x004A"
or "J"

```
FS1:\> mem ddf43018
Memory Address 00000000DDF43018 200 Bytes
  DDF43018: 4A 00 4A 00 4A 00 4A 00-4A 00 4A 00 4A 00 4A 00  *J.J.J.J.J.J.J.J.*
  DDF43028: 4A 00 4A 00 4A 00 4A 00-4A 00 4A 00 4A 00 4A 00  *J.J.J.J.J.J.J.J.*
  DDF43038: 4A 00 4A 00 4A 00 4A 00-4A 00 4A 00 4A 00 4A 00  *J.J.J.J.J.J.J.J.*
  DDF43048: 4A 00 4A 00 4A 00 4A 00-4A 00 4A 00 4A 00 4A 00  *J.J.J.J.J.J.J.J.*
```

tianocore

At the Shell prompt, type `FS1:\> unload FF`

```
FS1:\> unload ff
Unload - Handle [DDF49A18].  [y/n]?
y
Unload - Handle [DDF49A18] Result Success.
FS1:\> _
```

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

Type `Drivers` again to verify

```
FSOpen: Open '\' Success
[MyWizardDriver] Unload, clear buffer
[MyWizardDriver] Unload success
FSOpen: Open '\' Success
```

52

Lab 6: Verify Unload

At the Shell prompt, type `FS1:\> mem 0xDDF43018`

Observe the buffer is now NOT filled

```
FS1:\> mem ddf43018
Memory Address 00000000DDF43018 200 Bytes
  DDF43018: AF AF AF AF AF AF AF AF-AF AF AF AF AF AF AF AF   *................*
  DDF43028: AF AF AF AF AF AF AF AF-AF AF AF AF AF AF AF AF   *................*
  DDF43038: AF AF AF AF AF AF AF AF-AF AF AF AF AF AF AF AF   *................*
  DDF43048: AF AF AF AF AF AF AF AF-AF AF AF AF AF AF AF AF   *................*
```

Exit Simics `simics> stop, simics> quit`

LAB 7: ADD DRIVER TO THE PLATFORM

In this lab, you'll add the My Wizard Driver to the Platform Build.

Lab 7: Build the UEFI Driver

- **Open**
`edk2-platforms/Platform/Intel/SimicsOpenBoardPkg/BoardX58Ich10/OpenBoardPkg.fdf`

- **Add** the following in section [FV.DXEFV] and after the Shell.inf:

```
INF  ShellPkg/Application/Shell/Shell.inf

INF  MyPkg/MyWizardDriver/MyWizardDriver.inf
```

- **Save and close** the file `OpenBoardPkg.fdf`
- Optional - Update file `C:\fw\edk2-ws\edk2\MyPkg\MyWizardDriver.uni` for `FORM_SET_TITLE` and `FORM1_TITLE`, strings, Then Save. (Be Creative)

- **Build** the Simics BoardX58Ich10

```
$> cd C:\fw\edk2-ws\edk2-platforms\Platform\Intel
$> python build_bios.py -p BoardX58Ich10  -t VS20XX
```

Copy
`C:\fw\edk2-ws\Build\SimicsOpenBoardPkg\BoardX58Ich10\DEBUG_VS20XX\FV\BOARDX58ICH10.fd`
`To`
`%USERPROFILE%\AppData\Local\Programs\Simics\simics-qsp-x86-6.0.57\targets\qsp-x86\images`

Lab 7: Verify Driver Got Installed

Run the `qsp-modern-core` script from Windows Command Prompt :

```
$> .\simics targets/qsp-x86/qsp-modern-core.simics
simics> run
```

At the Shell prompt, type `Shell> drivers`

```
95 00000001 B - -  1  2 Super I/O Driver                LegacySioDxe
96 0000000A D - -  1  - PS/2 Keyboard Driver            Ps2KeyboardDxe
97 00000010 B - -  1  1 QEMU Video Driver               QemuVideoDxe
98 0000000A ? - -  -  - UEFI Sample Driver              MyWizardDriver
99 00002501 B X X  1  1 Intel(R) Gigabit 0.0.25.1       UndiDxe
Shell> _
```

Observe the handle is "98" as this slide example

At the Shell prompt, type `FS1:\> dmpstore -all MWD_NVData`

Observe new the NVRAM variable "`MWD_NVData`" was created and filled with the address of the buffer and the string "UEFI-Training-Class-MWD"
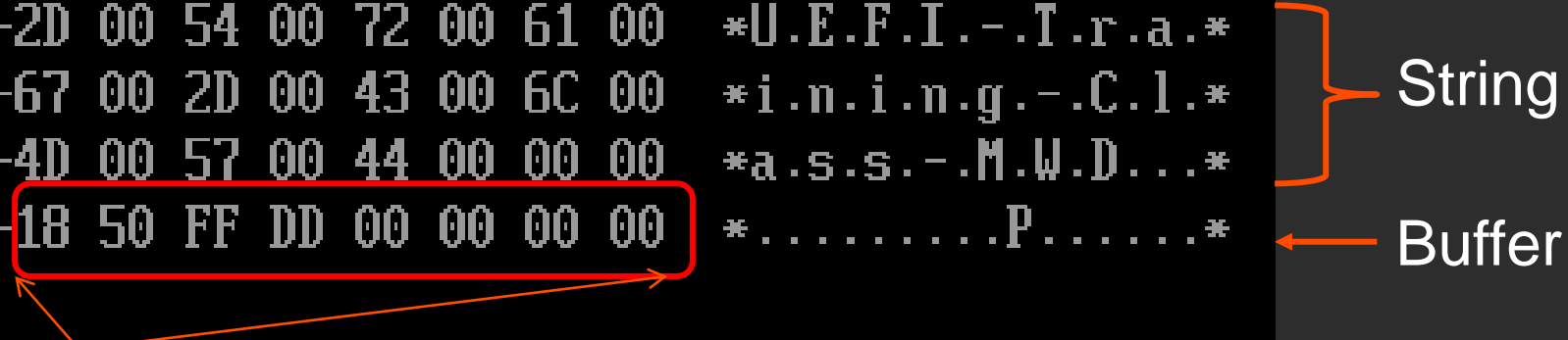
```
Shell> dmpstore -all MWD_NVDATA
Variable NV+BS '363729F9-35FC-40A6-AFC8-E8F54911F1D6:MWD_NVData' DataSize = 0x40
  00000000: 55 00 45 00 46 00 49 00-2D 00 54 00 72 00 61 00  *U.E.F.I.-.T.r.a.*
  00000010: 69 00 6E 00 69 00 6E 00-67 00 2D 00 43 00 6C 00  *i.n.i.n.g.-.C.l.*
  00000020: 61 00 73 00 73 00 2D 00-4D 00 57 00 44 00 00 00  *a.s.s.-.M.W.D...*
  00000030: 00 00 00 00 00 00 00 00-18 50 FF DD 00 00 00 00  *.........P......*
Shell> _
```

String

Buffer

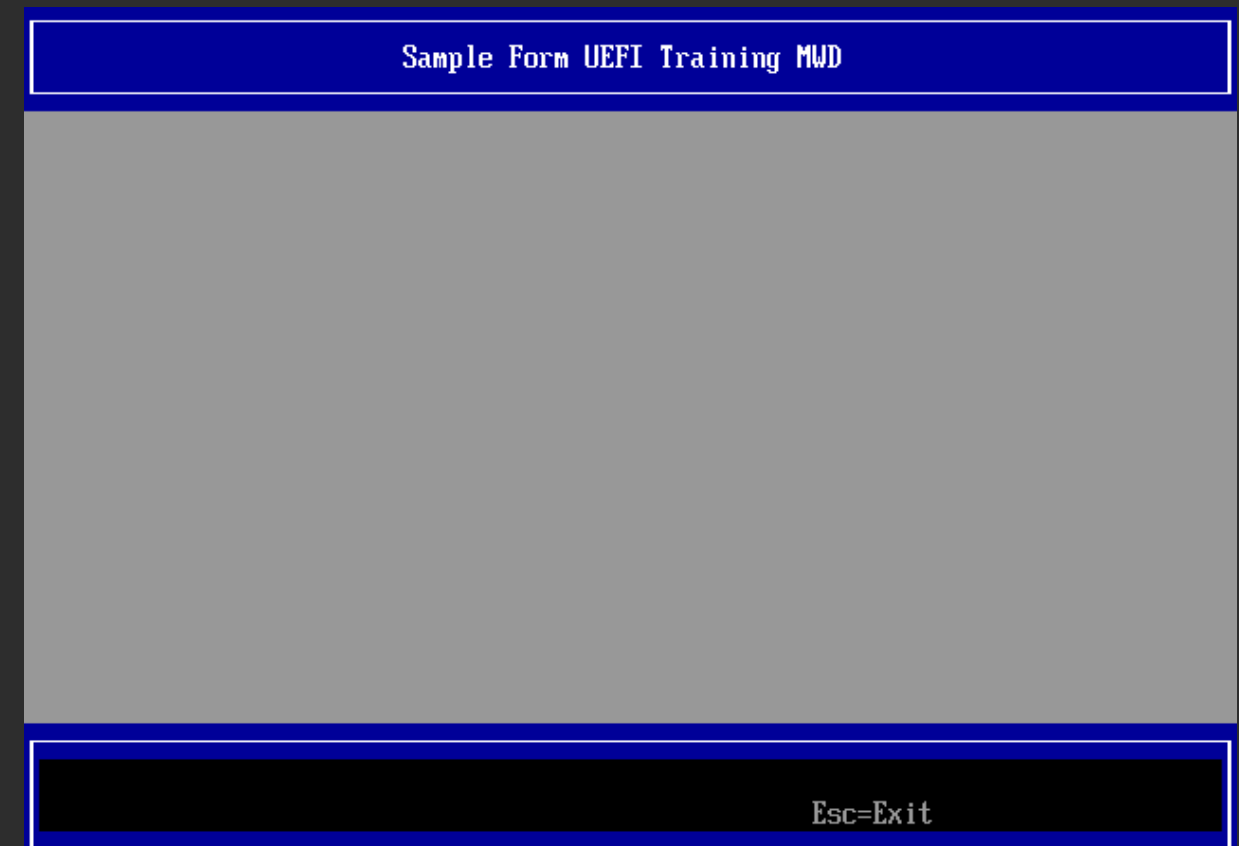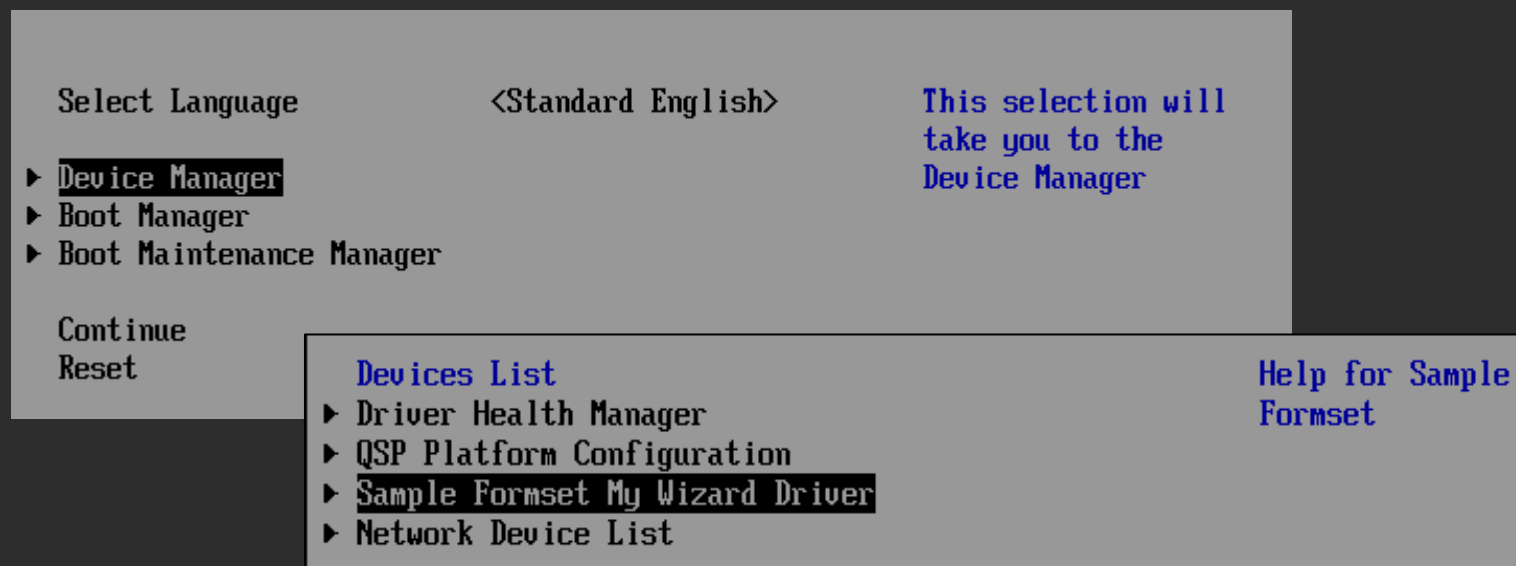Buffer address is: `00 00 DD FF 50 18`

At the Shell prompt, type `FS1:\> Mem DDFF5018` to Verify Buffer is still set to "J"s

Lab 7: Verify Driver Form Menu in Setup

At the Shell prompt, type `FS1:\>` Exit
This will exit back to setup, Then type "Escape", then Select "Device Manager"
and then "Sample Formset . . ."

This is the Form for the MyDriverWizard

This can be updated to get user data for configuration of your driver that then gets
stored in the NVRAM MWD_NVRam date

tianocore

Adding strings and forms to setup (HII)

Install produced protocols

Hardware initialization

Refer to the UEFI Drivers Writer's Guide for more tips– Pdf link

Summary

tianocore

- Compile a UEFI driver template created from UEFI Driver Wizard

- Test driver w/ Simics QSP Board using UEFI Shell 2.0

- Port code into the template driver

tianocore

Questions?

61

Return to Main Training Page

Return to Training Table of contents for next presentation link

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.