UEFI and EDK II Base Training

Lab and Reference Guide

Revised: 07/01/2013

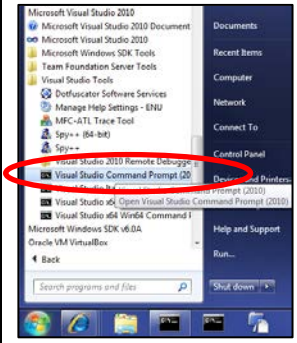
Lesson 1

UEFI Shell and EDK II Build Lab

1.1 Preparing for the Labs

Step	Action	
1	Training material is all on the Workstations under C:\Fw	
2	**\Presentations** **\Edk2** **\DriverWizard** **\LabSampleCode** **\Documentation**	Training Presentations & Lab work book Source code from TianoCore.org EDK II Project Installer for UEFI DriverWizard (. MSI) Solutions for EDK II Labs Help Documents

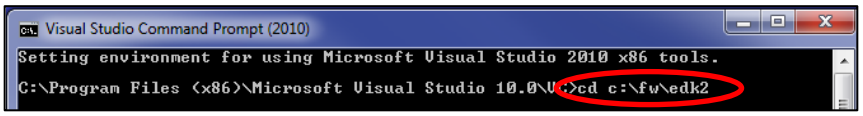
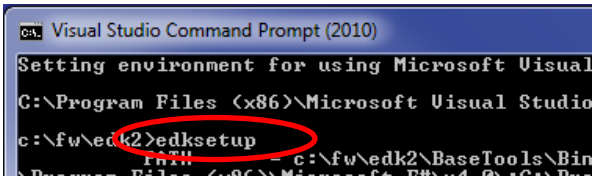
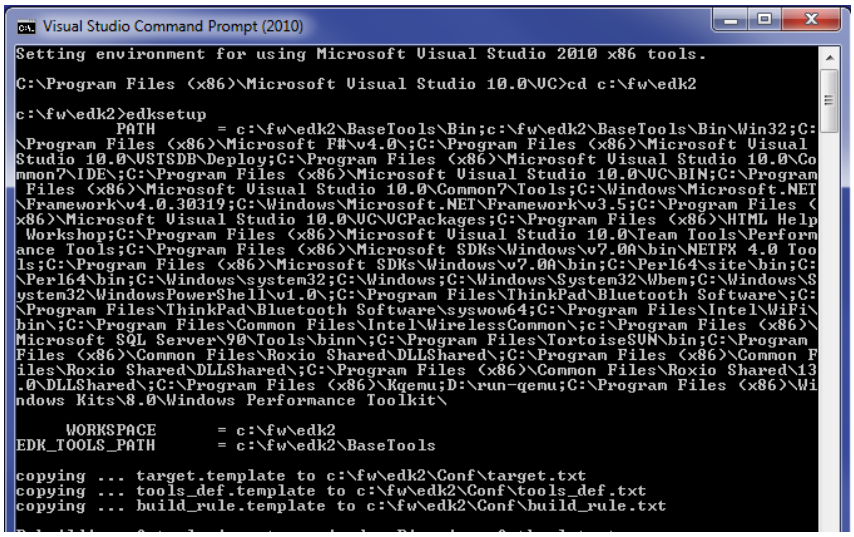
Selecting the Compiler Environment on Windows 7

Step	Action
3	**Open** Microsoft Visual Studio 2010
4	**Open** Visual Studio Tools
5	**Right-click** Visual Studio Command Prompt (2010)  ***Note***: The Visual Studio Command Prompt should be used instead of the standard Windows command prompt. The Visual Studio Command Prompt sets up the compiler environment for command line build.
6	**Pin** to the Task bar 

Preparing the EDKSETUP Command

Note: *You'll need to repeat this step each time you exit the Visual Studio Command Prompt. It is recommended that you keep your command prompt open during the training.*

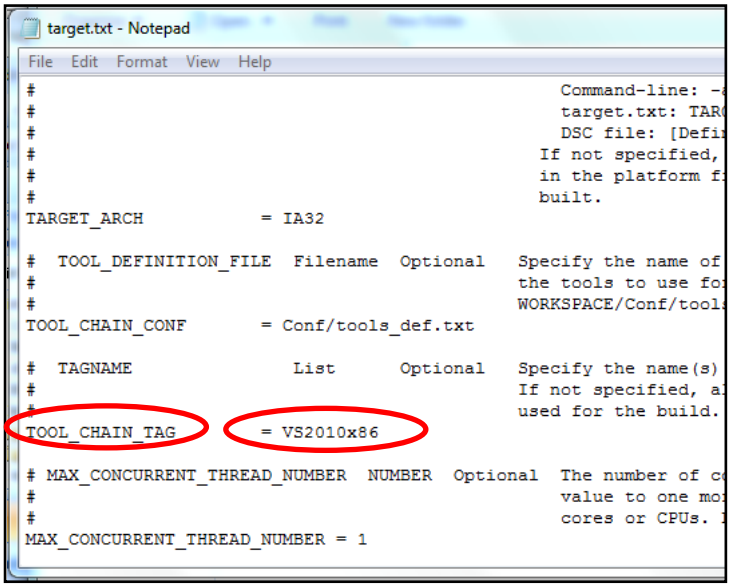
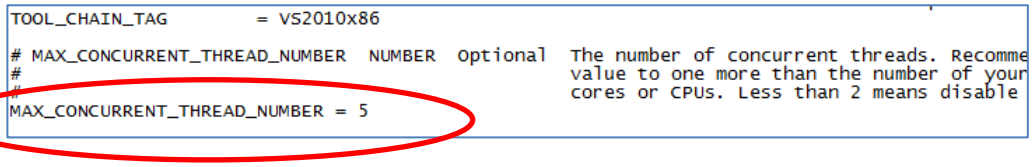
Step	Action
7	**Open** Visual Studio Command Prompt (2010 is used in this training guide)

Step	Action
8	**Type** CD `c:\fw\edk2` 
9	Press “Enter”
10	**Type** `edksetup` 
11	Press “Enter”  ***Note***: If you see “`!!! WARNING !!!...`”, don’t be alarmed. That can be ignored at this time.

Configuring Build Tools

Note: *You only need to edit Target.txt once. .*

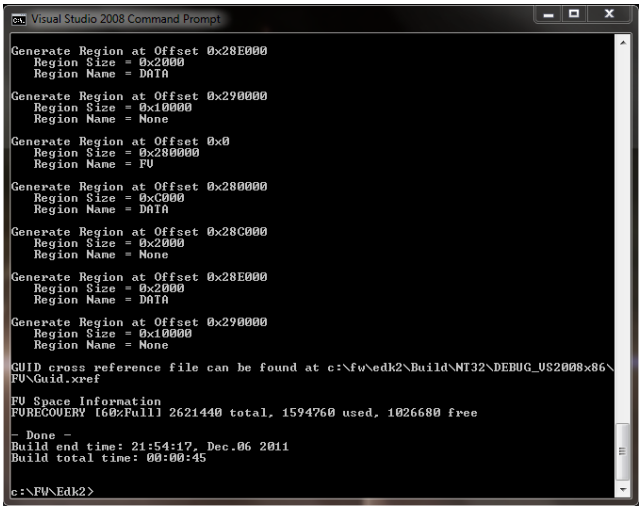
Step	Action
12	**Open** Notepad or other text editor that supports UNICODE
13	**Open** C:/fw/edk2/Conf/Target.txt

Step	Action
14	**Use** the Microsoft Windows and Visual Studio Matrix to **modify TOOL_CHAIN_TAG** to match your system  Example: for Windows 7 64 bit OS and Visual Studio 2010 modify the following in Target.txt From: **TOOL_CHAIN_TAG = MYTOOLS** to: **TOOL_CHAIN_TAG = VS2010x86**
15	Update the MAX_CONCURENT_THREAD_NUMBER By the number of processors on your laptop + 1. Example: most have Intel® Dual Core with Hyper threading which means 2 procs + 2 HT + 1 = 5. 
16	**Save** and **close** the text file
17	**Update** C:/fw/edk2/Conf/Tools_def.txt
18	DEFINE WINSDK_VERSION = **v7.0A** DEFINE WINSDK_BIN = C:\Program Files **(x86)**\Microsoft SDKs\Windows\DEF(WINSDK_VERSION)\bin 

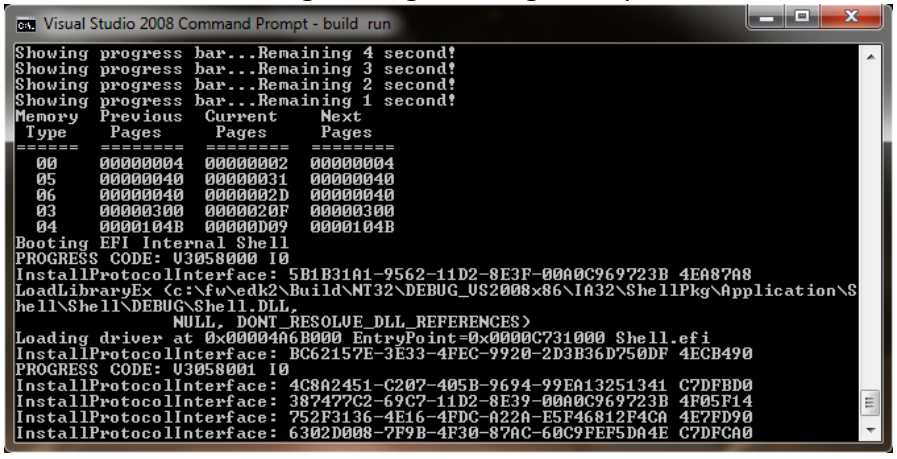
1.2 Building NT32

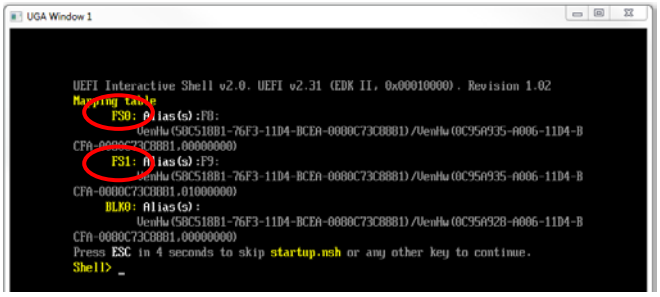
In this lab, you'll learn how to build the NT32 emulation.

Building NT32 (for the first time)

Step	Action
1	If you haven't performed the EDK II setup (see page 2), or downloaded and edited Target.txt (see page 23), do so first
2	In the Visual Studio Command Prompt**, type** `>build -D BUILD_NEW_SHELL` for the result below:  ***Note***: This can take upwards of five minutes

Viewing the NT32 Emulation Using the `Build Run` Command

Step	Action
3	In the Visual Studio Command Prompt**, type** `>build run`
	***Note**:* Notice the debug messages during startup 

Step	Action
4	A splash screen displays during the boot. Functions in Microsoft Windows handle output to the screen (console and GOP), input from the keyboard and storage on the file system. ***Note***: NT32 opens two output windows ('UGA Window 1' and 'UGA Window 2'). This behavior is defined by a Platform Configuration Database (PCD) token in the project. (You'll learn more about PCDs in Lessons 6 and 7)
5	NT32 loads the UEFI Shell under a 32-bit Microsoft Windows emulation environment. This example shows file systems **`fs0:`** and **`fs1:`**, which are mapped to a system directory by PCDs defined in the NT32 project. 

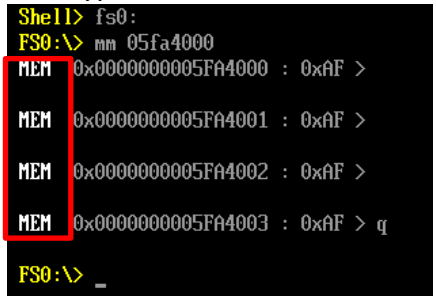
1.3 UEFI Shell Command Line Tools

In this lab, you'll learn how to use various UEFI shell command line tools, including the help and memory commands.

Memory Modify (MM) Command

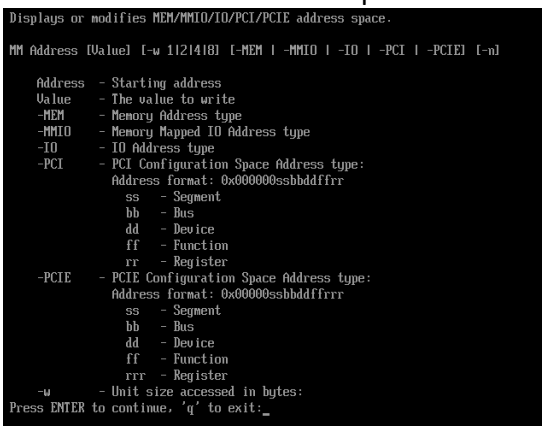
Displays or modifies memory, memory map I/O, I/O, and PCI address spaces.

Step	Action
1	If you haven't performed the EDK II setup (see page 2), or downloaded and edited Target.txt (see page 23), do so first
2	**Type** `build run`
3	Press "Enter"
4	**Type** `fs0:`
5	Press "Enter"

6	From the `2.0 Shell>` Prompt: If you have Windows 7, **type** `> MM 0005FA4000`
7	**Press** "Enter"
8	MEM type is the default  ***Note:*** By default, MM displays the contents of the memory address specified by the user. The command displays contents of the memory address entered on the command line. Having hit "enter", you proceeded to the next memory address. Entering a new value will change the contents of memory and 'q' will exit the program.

Help Command

Displays helps for the Memory Modify Command.

Step	Action
9	**Type** `mm -?`
10	**Press** "Enter"
	Here is the result of the Help Command 

Mem Command

– Displays the contents of the system or device memory.

When run without arguments, this command displays a summary of the system memory configuration.

Step	Action
11	**Type** > `mem`

Step	Action
12	**Press** "Enter"
13	Here is an example result of the **mem** command: 061EC170: AF AF AF AF AF AF AF AF-AF AF AF AF AF AF AF AF *.....* 061EC180: AF AF AF AF AF AF AF AF-AF AF AF AF AF AF AF AF *.....* Valid EFI Header at Address 00000000061EBF90 --- System: Table Structure size 00000048 revision 0002001F ConIn (000000000A3271F4) ConOut (0000000005373114) StdErr (000000000A3273A4) Runtime Services 00000000061EBF10 Boot Services 0000000000415C40 SAL System Table 0000000000000000 ACPI Table 0000000000000000 ACPI 2.0 Table 0000000000000000 MPS Table 0000000000000000 SMBIOS Table 000000000622F000 Shell> _

MemMap Command

Displays the memory map maintained by the UEFI environment.

Step	Action
1	**Type** > `memmap`
2	**Press** "Enter"
3	
	Here is an example result of the **MemMap** command: Available 00000000061C0000-000000000A1BFFFF 0000000000004000 000000000000000F MMIO 00000000021A0000-00000000021ABFFF 000000000000000C 8000000000000000 Reserved : 4 Pages (16,384) LoaderCode: 358 Pages (1,466,368) LoaderData: 23 Pages (94,208) BS_Code : 550 Pages (2,252,800) BS_Data : 3,895 Pages (15,953,920) RT_Code : 64 Pages (262,144) RT_Data : 64 Pages (262,144) ACPI Recl : 0 Pages (0) ACPI NVS : 0 Pages (0) MMIO : 12 Pages (49,152) Available : 27,810 Pages (113,909,760) Total Memory: 128 MB (134,266,880 Bytes) Shell> _

Drivers Command

Displays a list of currently loaded UEFI drivers.

Step	Action
4	**Type** > `drivers`
5	**Press** "Enter"

Step	Action
	Here is an example result of the **drivers** command: `5F 0000000A ? N N 0 0 Tcp Network Service Driver Fv(6D99E806-3D38-42C2-A095-5F4300BFD7DC)/FvFile(6D6963AB-906D-4A65-A7CA-BD40E5D6AF4D)` `60 0000000A ? N N 0 0 UDP Network Service Driver Fv(6D99E806-3D38-42C2-A095-5F4300BFD7DC)/FvFile(6D6963AB-906D-4A65-A7CA-BD40E5D6AF2B)` `61 0000000A ? N N 0 0 UEFI PXE Base Code Driver Fv(6D99E806-3D38-42C2-A095-5F4300BFD7DC)/FvFile(3B1DEAB5-C75D-442E-9238-8E2FFB62B0BB)` `62 0000000A ? N N 0 0 iSCSI Driver Fv(6D99E806-3D38-42C2-A095-5F4300BFD7DC)/FvFile(4579B72D-7EC4-4DD4-8486-083C86B182A7)` `64 0000000A ? N N 0 0 FAT File System Driver Fv(6D99E806-3D38-42C2-A095-5F4300BFD7DC)/FvFile(961578FE-B6B7-44C3-AF35-6BC705CD2B1F)` `Shell> _`

Devices Command

Displays a list of devices that UEFI drivers manage.

Step	Action
6	**Type** > `devices`
7	**Press** "Enter"
	Here is an example result of the **devices** command: `Shell> devices` `C T D` `T Y C I` `R P F A` `L E G G #P #D #C Device Name` `== = = = == == == ==========` `1C R - - 0 1 11 VenHw(58C518B1-76F3-11D4-BCEA-0080C73C8881)` `45 D - - 2 0 0 Primary Console Input Device` `46 D - - 2 0 0 Primary Console Output Device` `6C B - - 1 6 2 UGA Window 1` `6D B - - 1 6 2 UGA Window 2` `6E D - - 1 0 0 COM1` `6F D - - 1 0 0 COM2` `70 D - - 1 0 0 Bus Driver Console Window` `71 D - - 1 1 0 .` `72 D - - 1 1 0 ..\..\..\..\EdkShellBinPkg\Bin\Ia32\Apps` `73 D - X 1 2 0 Diskfile0` `74 D - - 1 0 0 a:RW;2880;512` `75 D - - 1 0 0 d:RO;307200;2048` `76 D - - 1 0 0 j:RW;262144;512` `Shell> _`

Devtree Command

Displays s tree of devices currently managed by UEFI drivers.

Step	Action
8	**Type >** `devtree`
9	**Press** "Enter"

Step	Action
	Here is an example result of the **devtree** command: Ctrl[6E] COM1 Ctrl[6F] COM2 Ctrl[70] Bus Driver Console Window Ctrl[71] . Ctrl[72] ..\..\..\..\EdkShellBinPkg\Bin\Ia32\Apps Ctrl[73] Diskfile0 Ctrl[74] a:RW;2880;512 Ctrl[75] d:RO;307200;2048 Ctrl[76] j:RW;262144;512 Ctrl[3B] VenHw(8614567D-35BE-4415-8D88-BD7D0C9C70C0) Ctrl[63] VenHw(6456ED61-3579-41C9-8A26-0A0BD62B78FC) Ctrl[66] VenHw(A04A27F4-DF00-4D42-B552-39511302113D) Ctrl[67] VenHw(B3F56470-6141-4621-8F19-704E577AA9E8) Ctrl[68] VenHw(3EBFA8E6-511D-4B5B-A95F-FB38260F1C27) Ctrl[69] VenHw(F76E0A70-B5ED-4C38-AC9A-E5F54BF16E34) Ctrl[6A] VenHw(847BC3FE-B974-446D-9449-5AD5412E993B) Ctrl[6B] VenHw(9E0C30BC-3F06-4BA6-8288-09179B855DBE) Shell> _

Driver Handle Database (DH) Command

Displays the device handles associated with UEFI drivers.

Step	Action
10	**Type** > dh
11	**Press** “Enter”
	Here is an example result of the **dh** command: 70: WinNTDriverIO DevicePath(D4-BD00-0080C73C8881,00000000)) 71: SimpleFileSystem WinNTDriverIO DevicePath(D4-BCFA-0080C73C8881,00000000)) 72: SimpleFileSystem WinNTDriverIO DevicePath(D4-BCFA-0080C73C8881,01000000)) 73: DiskIO BlockIO WinNTDriverIO DevicePath(D4-BCFA-0080C73C8881,00000000)) 74: WinNTDriverIO DevicePath(D4-BCFA-0080C73C8881,00000000)) 75: WinNTDriverIO DevicePath(D4-BCFA-0080C73C8881,01000000)) 76: WinNTDriverIO DevicePath(D4-BCFA-0080C73C8881,02000000)) 77: Shell ShellParameters SimpleTextOut UnknownDevice ImageDevicePath LoadedImage Shell> _

Load Command

Loads a UEFI driver into memory.

Step	Action
12	**Type** > `load -?`

Step	Action
13	**Press** "Enter"
	Here is an example result of the **load** with loading a UEFI Driver command: `Shell> fs0:` `FS0:\> load DebugPortDxe.efi` `Image 'FS0:\DebugPortDxe.efi' loaded at 606F000 - Success` `FS0:\> _`

Stall Command

Stalls the operation for a specified number of microseconds.

Step	Action
14	**Type** > `stall 10000000`
15	**Press** "Enter"
	Here is the result of the **stall** command: `FS0:\> stall 10000000` `FS0:\> _`
	***Note*: Keep your shell window open** for the next lab, 1.4

1.4 Writing UEFI Shell Scripts with EDK II

In this lab, you'll learn how to write a UEFI shell script with EDK II.

"Hello World" UEFI Shell Script

In this lab, you'll use the UEFI shell command `edit` to write a script.

Step	Action
1	If you haven't performed the EDK II setup (see page 2), or downloaded and edited Target.txt (see page 23), do so first
2	**Type** `build run`
3	Press "Enter"
4	**Type** `fs0:`
5	Press "Enter"
6	Using the shell window from Lab 1.3, **type** `edit HelloScript.nsh` `Shell> fs0:` `FS0:\> edit HelloScript.nsh_`

Step	Action
1	If you haven't performed the EDK II setup (see page 2), or downloaded and edited Target.txt (see page 23), do so first
2	**Type** `build run`
3	Press "Enter"
4	**Type** `fs0:`
5	Press "Enter"
7	Press "Enter"
8	**Type** `echo "Hello World"` 
9	**Press** "F2" to save
10	**Press "Enter"**
11	**Press** "F3" to exit the editor
12	In the shell, **type** `HelloScript` for the following result: 
13	Keep the NT32 Emulation Open for the next Lab section

Running the Date and Time Shell Scripts

Step	Action
14	**Locate** Script1.nsh and Script2.nsh in: C:\FW\LabSampleCode\ShellScripts
15	**Copy** the files Script1.nsh and Script2.nsh to: c:\Fw\edk2\Build\NT32\DEBUG_VS20XXx86\IA32 ***Note***: The two colored XXs represent your version of Visual Studio
16	From the FS0: Shell Prompt
17	**Type** `script1`

Step	Action
18	**Press** "Enter". **Keep the prompt open**. This is the results when you run the shell script successfully: `FS0:\> script1` `FS0:\> script2.nsh` `FS0:\> time > Mytime.log` `FS0:\> for %a run (3 1 -1)` `FS0:\>  echo %a counting down` `3 counting down` `FS0:\> endfor` `FS0:\> for %a run (3 1 -1)` `FS0:\>  echo %a counting down` `2 counting down` `FS0:\> endfor` `FS0:\> for %a run (3 1 -1)` `FS0:\>  echo %a counting down` `1 counting down` `FS0:\> endfor` `FS0:\> for %a run (3 1 -1)` `FS0:\> if exist %cwd%Mytime.log then` `FS0:\>  type Mytime.log` `12:40:37 (UTC-08:00)` `FS0:\> endif` `FS0:\> echo "Thank you. ByeBye:) "` `Thank you. ByeBye:)` `FS0:\> _`

Editing the Date and Time Shell Scripts

Step	Action
19	At the shell prompt (`Shell FS0:\>`), type `edit Script1.nsh`
20	**Delete** the # before the `echo -off` command
21	**Press** "F2" to save
22	Press "Enter"
23	**Press** "F3" to exit `Script1.nsh`
24	**Type** `script1`
25	Press "Enter" for the following result `FS0:\> script1` `FS0:\> echo -off` `3 counting down` `2 counting down` `1 counting down` `12:41:36 (UTC-08:00)` `Thank you. ByeBye:)` `FS0:\> _`
26	**Type** `Reset` to exit the NT32 Emulation.

LESSON 6

EDK II PLATFORM CONFIGURATION DATABASE (PCD)

6.1 Changing a PCD Value

In this lab, you'll learn how to change a Platform Configuration Database (PCD) value. In Figure 1, notice that the names of the two windows are different. The names were modified by changing the PCD value.

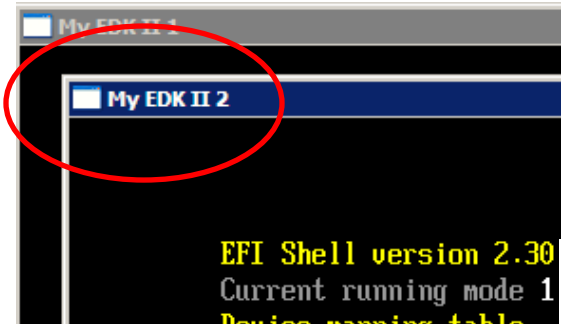
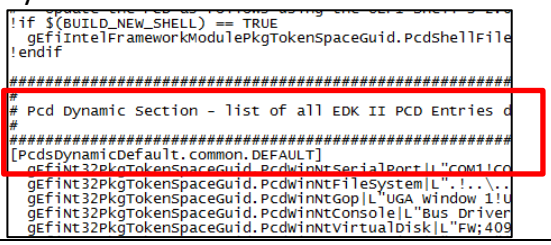
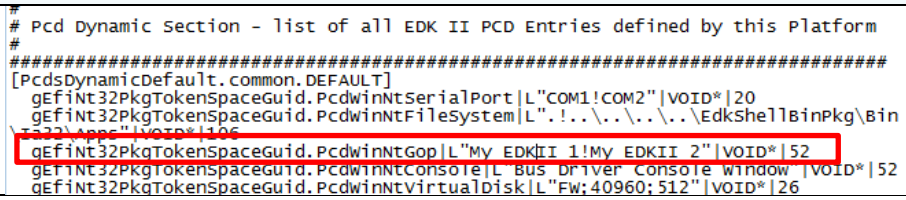
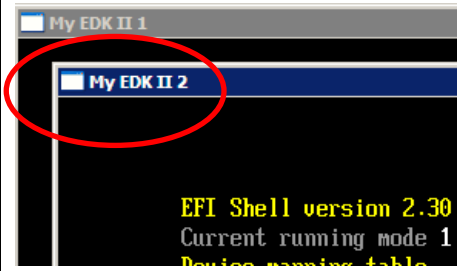


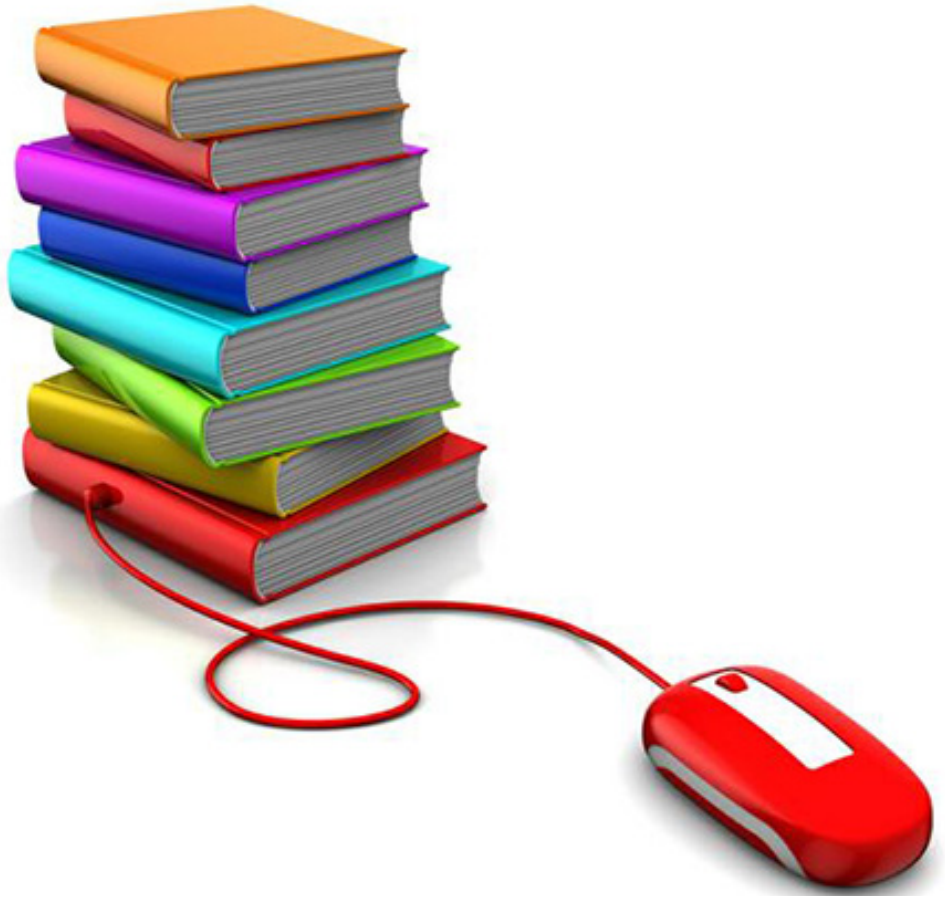
Figure 1 *Changing the PCD value modifies the window names*

Step	Action
1	If you haven't performed the EDK II setup (see page 2), or downloaded and edited Target.txt (see page 23), do so first
2	**Open** C:\Fw\edk2\Nt32Pkg\Nt32Pkg.dsc
3	**Locate** `[PcdsDynamicDefault.common.DEFAULT]` in the dynamic PCD section 
4	**Change** the PCD in the image below to the following string: `gEfiNt32PkgTokenSpaceGuid.PcdWinNtGop|L"My EDK II 1!My EDK II 2"|VOID*|52` 
5	Save and close the `Nt32Pkg.dsc` file
6	In the Visual Studio Command Prompt, **type** `CD C:\FW\edk2`
7	**Press** "Enter"
8	**Type** `edksetup`
9	**Press** "Enter"
10	**Type** `Build -D BUILD_NEW_SHELL`

Step	Action
11	**Press** "Enter"
12	**Type** `Build Run`
13	Press "Enter"
14	The new NT32 window names take effect when NT32 starts. 

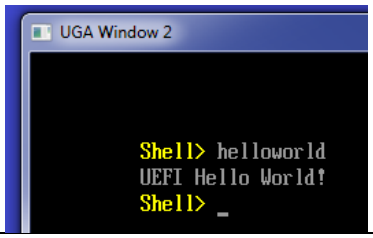
LESSON 7

UEFI Application Writers Lab (for UEFI Shell 2.0)



7.1 Writing UEFI Applications with PCDs

In this lab, you'll learn how to write UEFI applications with PCDs.

Step	Action
1	If you haven't downloaded the UEFI training materials , run edksetup or edited Target.txt (see page 23), do so first
2	In the Visual Studio Command Prompt, t**ype** `cd c:\fw\edk2`
3	Press "Enter"
4	**Type** `build run`
5	Press "Enter" to start the NT32 emulation.
6	Once NT32 loads the UEFI Shell, **type** `helloworld`
7	Press "Enter". Observe the output of the **`HelloWorld`** program. 
8	**Type** `Reset`
9	Press "Enter"

Modifying HelloWorld Behavior

In this part of the lab, you'll learn how to modify HelloWorld behavior. You're given HelloWorld running as-is (*Figure 2*) and are responsible for changing it. Using PCD values, you can change elements of this program instead of changing source code in the C file allowing for changes using the DSC file.

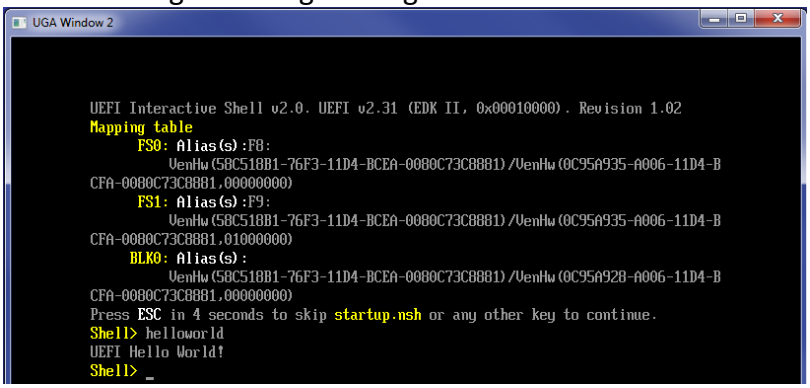
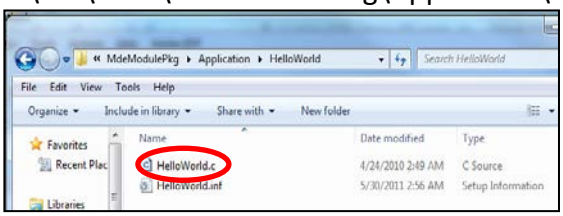
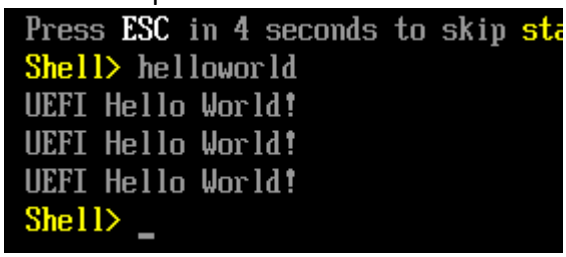


Figure 2 *HelloWorld running as-is*

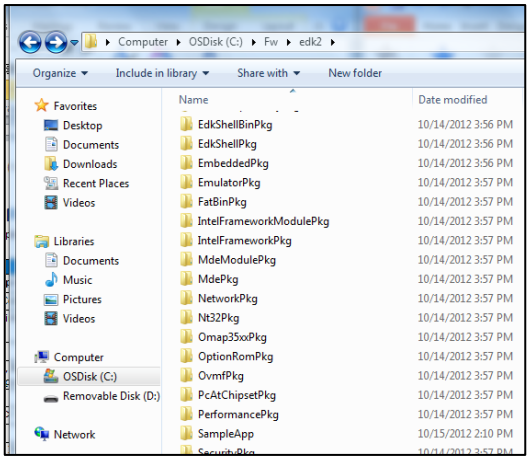
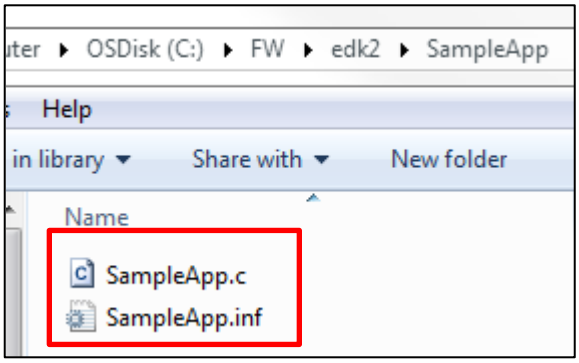
Step	Action

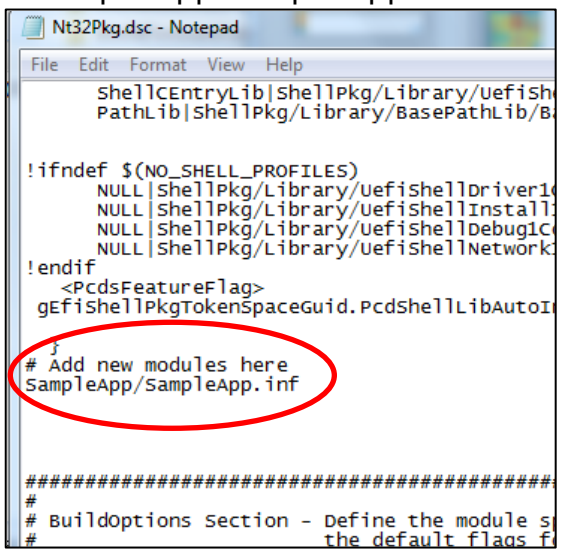
10	**Locate** and **open** C:\FW\edk2\MdeModulePkg\Application\HelloWorld\HelloWorld.c 
11	**Notice** the PCD values referenced in the source code. `PcdHelloWorldPrintEnable` `PcdHelloWorldPrintTimes` `PcdHelloWorldPrintString` *Note*: In the next few steps, you'll change a PCD value which prints the string "`UEFI HelloWorld!!`" three times without changing any C Code. **HelloWorld.c** `EFI_STATUS` `EFIAPI` `UefiMain (` `  IN EFI_HANDLE  ImageHandle,` `  IN EFI_SYSTEM_TABLE  *SystemTable` `  )` `{` `  UINT32 Index;` `  Index = 0;` `  // Three PCD type (FeatureFlag, UINT32` `  // and String) are used as the sample` `  if (FeaturePcdGet (PcdHelloWorldPrintEnable)) {` `  for (Index = 0; Index < PcdGet32  (PcdHelloWorldPrintTimes);` `  Index ++) {` `  // Use UefiLib Print API to print` `  // string to UEFI console` `  Print ((CHAR16*)PcdGetPtr (PcdHelloWorldPrintString));` `  }` `  }` `  return EFI_SUCCESS;` `}`
12	**Open** C:\Fw\edk2\Nt32Pkg\Nt32Pkg.dsc
13	**Find** `[PcdsFixedAtBuild]`
14	After `[PcdsFixedAtBuild]`, **add the following line:** `gEfiMdeModulePkgTokenSpaceGuid.PcdHelloWorldPrintTimes|3` `[PcdsFixedAtBuild]` `  gEfiMdeModulePkgTokenSpaceGuid.PcdHelloWorldPrintTimes|3` `  gEfiMdeModulePkgTokenSpaceGuid.PcdMaxSizePopulateCapsule|0x0`
15	Save the `Nt32Pkg.dsc` file and exit the text editor
16	To rebuild in the Visual Studio Command Prompt, **type** `build -D BUILD_NEW_SHELL` **(Remember "_" in the switch)**
17	**Press** "Enter"

18	To lauch NT32, **type** `build run`
19	Press “Enter”
20	**Type** `Helloworld` to run the modified `helloworld` program, which now prints the `UEFI Hello World!` string three times:  ***Note***: Ask yourself: “How would you change the string ‘UEFI Hello World’ using the PCDs?”
21	At the Shell 2.0 prompt, **type** `reset`
22	**Press** “Enter” to close the NT32 Emulation.

7.2 Writing Simple UEFI Applications

In this lab, you'll learn how to write simple UEFI applications.

Step	Action
1.	**Create** a folder called **SampleApp** in the C:\fw\edk2 workspace 
2	Now**, locate** and **open**: C:\FW\LabSampleCode\SampleApp
3	**Copy** the following files: 1. SampleApp.c 2. SampleApp.inf
4	**Paste** them into into the **SampleApp** folder you created in the C:\Fw\edk2 workspace 
5	**Open** SampleApp.inf
6	**Replace** the XXXXXXXXs with the following with text in red:

Step	Action
	[Defines] INF_VERSION = 0x00010005 BASE_NAME = **SampleApp** FILE_GUID = (retrieve GUID from GuidGen.com) MODULE_TYPE = **UEFI_APPLICATION** VERSION_STRING = 1.0 ENTRY_POINT = UefiMain [Sources] **SampleApp.c** [Packages] **MdePkg/MdePkg.dec** [LibraryClasses] **UefiApplicationEntryPoint**
7	**Save** the file and leave it open.
8	**Open** C:\Fw\edk2\Nt32Pkg\Nt32Pkg.dsc
9	**Add** the followig above the [BuildOptions] section of Nt32Pkg.dsc: SampleApp/SampleApp.inf 
10	**Save** the file and exit the editor.
11	**Open** C:\Fw\edk2\SampleApp\SampleApp.c
12	**Add** the following include statements near the top of **SampleApp.c**:
	#include <Uefi.h> #include <Library/UefiApplicationEntryPoint.h>

Step	Action
	WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED. **/ #include <Uefi.h> #include <Library/UefiApplicationEntryPoint.h> /** as the real entry point for the application. @param[in] ImageHandle The firmware allocated handle for the EFI image. @param[in] SystemTable A pointer to the EFI System Table.
13	**Save** the file.

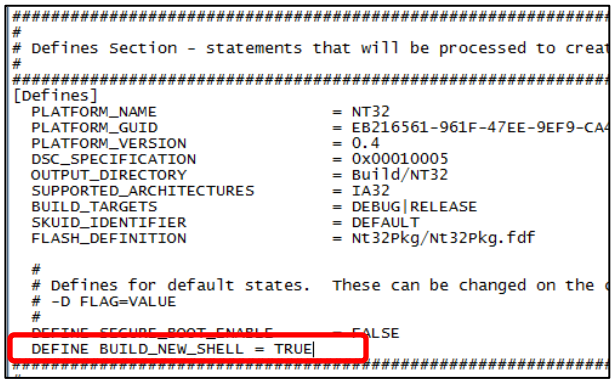
Compiling the NT32 Emulation

Step	Action
14	In the Visual Studio Command Prompt**, type** `build -D BUILD_NEW_SHELL`
15	Press "Enter"
16	**Type** `build run`
17	Press "Enter" to start the NT32 Emulation
18	Once NT32 loads the UEFI Shell, **type** `SampleApp`
19	Press "Enter" to run the sample application. The program will exit back to the shell without any errors. `Shell> sampleapp` `Shell> _`
20	**Type** `reset`
21	Press "Enter" to exit NT32.
22	If BUILD Errors: Invoke the following Batch file from Windows Explorer: C:\FW\LabSampleCode\LabSolutions\Lesson7.2\Lesson72.bat

Click the "link for solution files for common build errors:

Lesson 7.2: Build Errors for Compiling

7.2.1 Compiling NT32 without a Command Line Switch

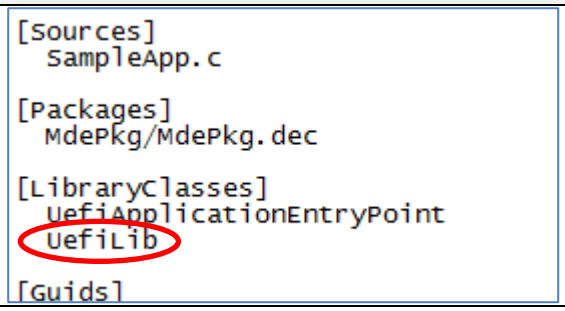
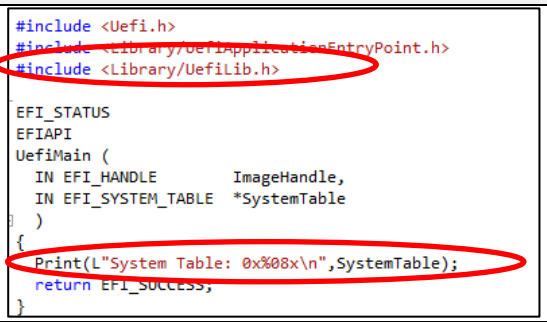
Step	Action
23	**Open** C:\Fw\edk2\Nt32Pkg\Nt32Pkg.dsc
24	**Change** `DEFINE BUILD_NEW_SHELL = FALSE to "TRUE"` 
25	**Save** C:\Fw\edk2\Nt32Pkg\Nt32Pkg.dsc and exit the editor.
26	In the Visual Studio Command Prompt, **type** `build`
27	Press "Enter
28	**Type** `build run`
29	Press "Enter" to start the NT32 Emulation.

7.3 Adding Functionality to UEFI Applications

In this lab, you'll learn how to add functionality to UEFI applications.
Add code to print the hex address of the EFI System Table pointer to the console.

Why	**To add some "C" code that will display the EFI System Table because it is the pointer to all the other services available to access.**
How:	**Search MdePkg Document With Libraries.chm for the library call to `print`**

Modifying .C and .INF Files

Step	Action
1	**Open** C:\Fw\edk2\SampleApp\SampleApp.inf
2	**Add the following** to `[LibraryClasses]`as shown below:
	`UefiLib`
	
3	**Save** the file and exit the editor
4	**Open** C:\Fw\edk2\SampleApp\SampleApp.c
5	**Add** the following text in the locations shown below:
	`#include <Library/UefiLib.h>`
	`Print(L"System Table: 0x%08x\n", SystemTable);`
	
6	**Save** the file
7	In the Visual Studio Command Prompct, **type** `build`

Step	Action
8	Press “Enter” to rebuild the NT32 project
9	**Type** `build run`
10	Press “Enter” to start the NT32 Emulation
11	At the UEFI Shell prompt, **type** `SampleApp` Press ESC in 4 seconds to skip st Shell> sampleapp System Table: 0x062BBF90 Shell> _
12	**Press** “Enter” to see the changes in the application behavior
13	**Test** with the Shell “Mem” Command. EFI Header will equal the EFI System Table Valid EFI Header at Address 00000000062BBF90 ---------- System: Table Structure size 00000048 revision 0002001F ConIn (0000000001FC71F4) ConOut (0000000005444114) StdErr (0000000001FC73A4) Runtime Services 00000000062BBF10 Boot Services 00000000004D5C40 SAL System Table 0000000000000000 ACPI Table 0000000000000000 ACPI 2.0 Table 0000000000000000 MPS Table 0000000000000000 SMBIOS Table 00000000062FF000 Shell> sampleapp System Table: 0x062BBF90 Shell> _
14	**Type** `reset`
15	Press “Enter” to exit the NT32 Emulation

7.4 Writing EFI Code for Waiting for an Event

In this lab, you'll learn how to locate code and .chm files to help write EFI code for waiting for an event:

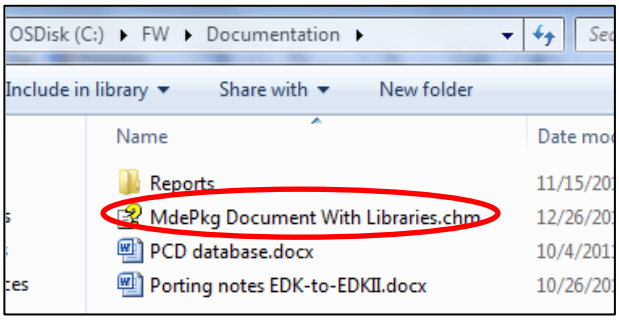
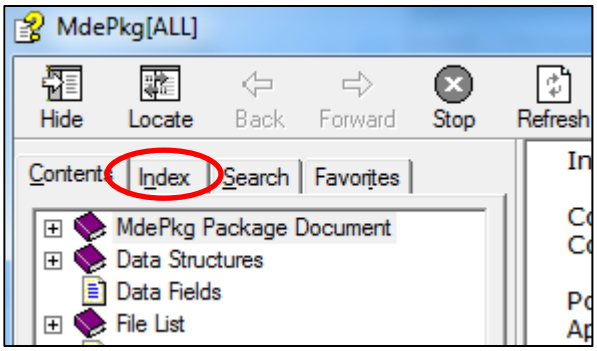
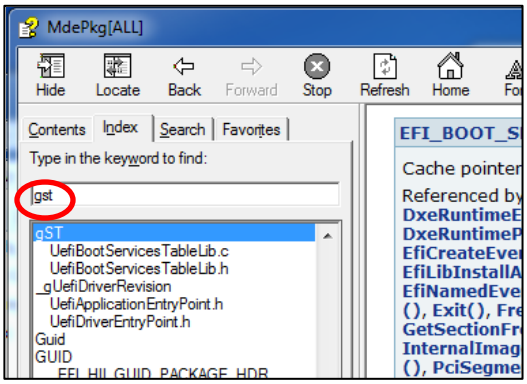
WaitForEvent and WaitForKey Locations

- ✓ MdePkg Document With Libraries.chm
- ✓ UEFI Specification
- ✓ Search Work Space: MdePkg/Library/UefiLib/Console.c (~line 569)

Console.c (line ~569):

```
//
if (Status != EFI_NOT_READY) {
  continue;
}
gBS->WaitForEvent (1, &gST->ConIn->WaitForKey, &EventIndex);
```

Step	Action
1	**Open** C:\Fw\edk2\SampleApp.c
2	**Add (copy and paste)** the following text in the location show below:
	`UINTN EventIndex;`
	`Print(L"\n\n Press any Key to continue : \n");` `gBS->WaitForEvent (1, &gST->ConIn->WaitForKey, &EventIndex);`
	`{` `  UINTN EventIndex;` `  Print(L"System Table: 0x%08x\n",SystemTable);` `  Print(L"\n\n Press any Key to continue : \n");` `  gBS->WaitForEvent (1, &gST->ConIn->WaitForKey, &EventIndex);` `  return EFI_SUCCESS;` `}` ***Note**: This won't compile because gBS and gST are not defined.*
3	**Save** the file, but keep it open

Step	Action
4	**Open** C:\FW\Documentation\MdePkg Document With Libraries 
5	**Click** Index 
6	**Search** for gBS and gST separately to get an idea of where they are located.  ***Note**: Notice that they are located in the* `UefiBootServicesTableLib.h`
7	**Return** to the SampleApp.c file that you left open
8	In the SampleApp.c file **add** the following at the top with the other #include statements `#include <Library/UefiBootServicesTableLib.h>`
9	**Save** the file
10	In the Visual Studio Command Prompt, **type** `build`
11	**Type** `build run` to start the NT32 Emulation
12	**Press** "Enter"
13	At the Shell 2.0 prompt, **type** `SampleApp`

Step	Action
14	Press “Enter” to run the SampleApp program (example below): 
15	At the Shell 2.0 prompt, **type** `reset`
16	**Press** “Enter” to close the NT32 Emulation.

7.5 Creating a Simple Typewriter Function

In this lab, you'll learn how to create a simple typewriter function that retrieves the keys you type and subsequently prints each one back to the console.

Requirements

- ✓ Retrieve keys entered from keyboard
- ✓ Print back each key entered to the console
- ✓ To exit, press "." and then <Enter> key

Process

1. Add a Loop using *WaitForEvent* with *WaitForKey*
2. Use the *ReadKeyStroke* function from *ConIn*
3. Print back each key to console
4. Exit when "." character followed by a <Enter> key

Step	Action
1	**Open** C:\fw\edk2\SampleApp\SampleApp.c
2	**Add (copy and paste)** the following code to the `#include` section:
	`#include <Library/BaseMemoryLib.h>` `#define CHAR_DOT 0x002E // '.' in Unicode`
	`#include <Library/UefiLib.h>` `#include <Library/UefiBootServicesTableLib.h>` `#include <Library/BaseMemoryLib.h>` `#define CHAR_DOT 0x002E // '.' in Unicode`
3	**Locate** the function `UefiMain ()`. **Add (copy and paste)** the following variable definitions to the function:
	`BOOLEAN ExitLoop;` `EFI_INPUT_KEY Key;`

Step	Action

```
EFI_STATUS
EFIAPI
UefiMain (
  IN EFI_HANDLE        ImageHandle,
  IN EFI_SYSTEM_TABLE  *SystemTable
  )
{
    UINTN          EventIndex;
    BOOLEAN        ExitLoop;
    EFI_INPUT_KEY      Key;

    Print(L"System Table: 0x%08x\n", SystemTable);
    Print(L"\n\n Press any Key to continue : \n");
```

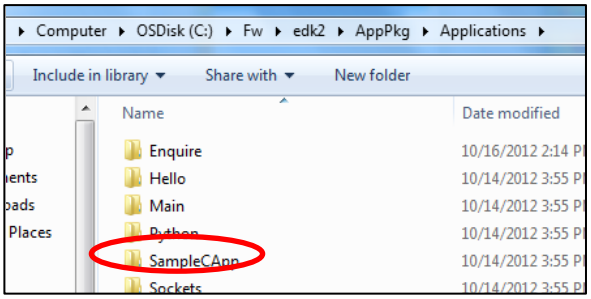
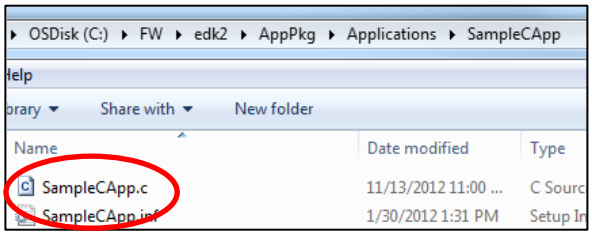
4 **Add (copy and paste)** the following block of code above the `return EFI_SUCCESS;` statement at the end of the function:

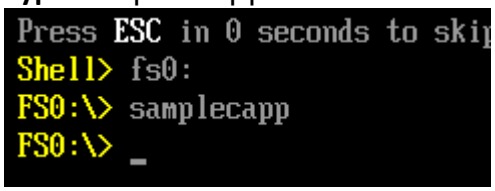
```
      Print(L"Enter text. Include a dot ('.') in a sentence then
<Enter> to exit:\n\n");
      ZeroMem (&Key, sizeof (EFI_INPUT_KEY));
      gST->ConIn->ReadKeyStroke (gST->ConIn, &Key);
      ExitLoop = FALSE;
      do {
            gBS->WaitForEvent (1, &gST->ConIn->WaitForKey,
&EventIndex);
            gST->ConIn->ReadKeyStroke (gST->ConIn, &Key);
            Print(L"%c", Key.UnicodeChar);
            if (Key.UnicodeChar == CHAR_DOT){
                  ExitLoop = TRUE;
            }
      } while (!(Key.UnicodeChar == CHAR_LINEFEED ||
                  Key.UnicodeChar == CHAR_CARRIAGE_RETURN) ||
                      !(ExitLoop) );

      Print(L"\n");
```

Step	Action
	`` UINTN EventIndex; BOOLEAN ExitLoop; EFI_INPUT_KEY Key; Print(L"System Table: 0x%08x\n", SystemTable); Print(L"\n\n Press any Key to continue : \n"); gBS->WaitForEvent (1, &gST->ConIn->WaitForKey, &EventIndex); Print(L"Enter text. Include a dot ('.') in a sentence then <Enter> to exit:\n\n"); ZeroMem (&Key, sizeof (EFI_INPUT_KEY)); gST->ConIn->ReadKeyStroke (gST->ConIn, &Key); ExitLoop = FALSE; do { gBS->WaitForEvent (1, &gST->ConIn->WaitForKey, &EventIndex); gST->ConIn->ReadKeyStroke (gST->ConIn, &Key); Print(L"%c", Key.UnicodeChar); if (Key.UnicodeChar == CHAR_DOT){ ExitLoop = TRUE; } } while (!(Key.UnicodeChar == CHAR_LINEFEED || Key.UnicodeChar == CHAR_CARRIAGE_RETURN) || !(ExitLoop)); Print(L"\n"); return EFI_SUCCESS; } ```
5	**Save** and **close** the file
6	In the Visual Studio Command Prompt, **type** `build`
7	Press “Enter” to compile the NT32 project
8	**Type** `build run`
9	Press “Enter” to start the NT32 Emulation
10	At the Shell 2.0 prompt**, type** `SampleApp`
11	Press “Enter” to run and test the application (see below). ``` Shell> sampleapp System Table: 0x061CBF90 Press any Key to continue : Enter text. Include a dot ('.') in a sentence then <Enter> to exit: This is text from the type writer function. Shell> _ ```
12	At the Shell 2.0 prompt, **type** `reset`
13	**Press** “Enter” to close the NT32 Emulation.

7.6 Writing UEFI Applications with EADK

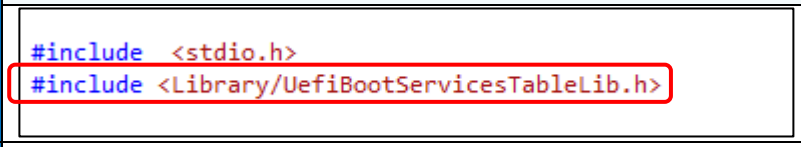
In this lab, you'll write an application with the same functionality as **SampleApp.c** using LibC from the EDK II Application Development Kit (EADK).

Step	Action
1	**Create** a folder called **SampleCApp** in C:\Fw\edk2\AppPkg\Applications 
2	**Locate** and **open**: C:\Fw\LabSampleCode\SampleCApp
3	**Copy** the files from the c:\fw\LabSamplecode\SampleCApp folder to C:\Fw\edk2\AppPkg\Applications\SampleCApp 
4	**Open** C:\Fw\edk2\AppPkg\AppPkg.dsc
5	**Add** the following at the end of the Components section:
	`AppPkg/Applications/SampleCApp/SampleCApp.inf`
	`[Components]` `#### Sample Applications.` `  AppPkg/Applications/Hello/Hello.inf  # No LibC in` `  AppPkg/Applications/Main/Main.inf  # Simple invo` `  AppPkg/Applications/Enquire/Enquire.inf  #` `#  AppPkg/Applications/SampleCApp/SampleCApp.inf` `#### After extracting the Python distribution, un-comment` `#  AppPkg/Applications/Python/PythonCore.inf` `AppPkg/Applications/SampleCApp/SampleCApp.inf`
6	**Save** and **close** the file
7	Open the Visual Studio Command Prompt and **type** Cd

Step	Action
	`C:\fw\edk2`
8	**Press** "Enter" to change directories
9	In the Visual Studio Command Prompt, **type** `edksetup`
10	Press "Enter"
11	**Type** `build -p AppPkg\AppPkg.dsc`
12	**Press** "Enter" to build SampleCApp without rebuilding the entire NT32 project. ***Note**: The build takes a few minutes to complete*
13	Next, **type** the following to copy the compiled application into a directory used by the NT32 Emulation: `copy build\AppPkg\DEBUG_VS2010x86\IA32\SampleCApp.efi build\NT32\DEBUG_VS2010x86\IA32` ***Caution**: The line of code is one line without breaks. Also ensure that you edit the line of code to match your version of Visual Studio*
14	**Press** "Enter" to copy the .EFI executable
15	**Keep** your Visual Studio Command Prompt open
16	In the open Visual Studio Command Prompt**,**
17	**Type** `build run`
18	**Press** "Enter" to start the NT32 Emulation
19	At the Shell 2.0 prompt, **type** `fs0:`
20	**Press** "Enter" to change directories
21	**Type** `SampleCApp`  Note: your Sample application does not do anything yet
22	**Press** "Enter" to run the new version of `SampleCApp`

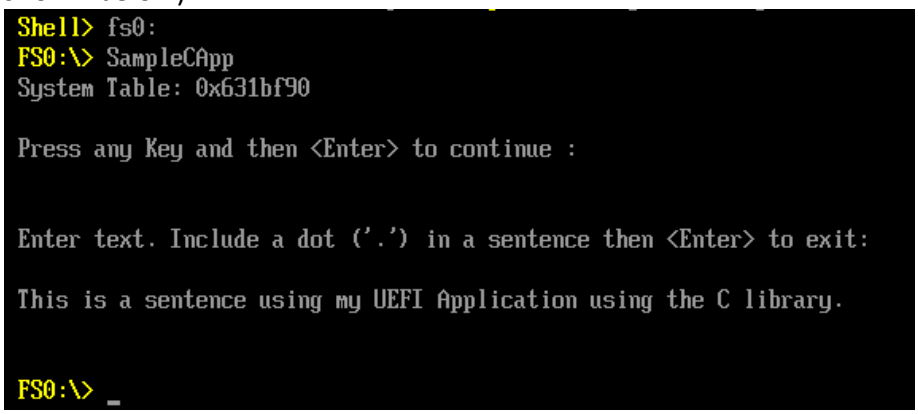
7.7 Adding Functionality to SampleCApp

In this lab, you'll add functionality to SampleCApp the same as in Lab 7.5. This lab will use EADK libraries so the coding style is similar to standard C.

Step	Action
1	**Open** C:\Fw\edk2\AppPkg\Applications\SampleCApp\SampleCApp.c
2	**Add the following** to the `include` section at the top of the file
	`#include <Library/UefiBootServicesTableLib.h>`
	
3	**Locate** the `main()` function.
4	**Add (copy and paste)** the following code to the `main()` function:

```
   char c;

   printf("System Table: %p \n", gST) ;

   puts("\nPress any Key and then <Enter> to
continue :  \n");
   c=(char)getchar();

   puts ("Enter text. Include a dot ('.') in a
sentence then <Enter> to exit:\n");
  do {
      c=(char)getchar();
     } while (c != '.');

  puts ("\n");
```

Step	Action
	`` int EFIAPI main (IN int Argc, IN char **Argv) { char c; printf("System Table: %p \n", gST) ; puts("\nPress any Key and then <Enter> to continue : \n"); c=(char)getchar(); puts ("Enter text. Include a dot ('.') in a sentence then <Enter> to exit:\n"); do { c=(char)getchar(); } while (c != '.'); puts ("\n"); return 0; } ``` ***Note***: This code perfoms the same functions as the previous SampleApp code. The code formatting follows a standard ANSI C model by using the EADK libraries.
5	**Save** and **close** the file
6	In the Visual Studio command prompt**, type** `cd C:\fw\edk2`
7	**Press** "Enter" to change directories ***Note***: If you've closed your Visual Studio Command Prompt prior to these steps, you'll need to rerun edksetup by typing `edksetup` and pressing "Enter"
8	**Type** `build -p AppPkg\AppPkg.dsc`
9	**Press** "Enter" to build SampleCApp without rebuilding the entire NT32 project. ***Note***: *The build takes a few minutes to complete*
10	**Type** `copy build\AppPkg\DEBUG_VS2010x86\IA32\SampleCApp.efi build\NT32\DEBUG_VS2010x86\IA32` to copy the compiled application into a directory used by the NT32 Emulation. ***Caution***: *The line of code is one line without breaks. Also ensure that you edit the line of code to match your version of Visual Studio*
11	**Press** "Enter" to copy the .EFI executable
12	**Type** `build run`
13	**Press** "Enter" to start the NT32 Emulation
14	At the Shell 2.0 prompt, **type** `fs0:`
15	**Press** "Enter" to switch directories.
16	**Type** `SampleCApp`

Step	Action
17	**Press** "Enter" to run the new version of **`SampleCApp`** (example shown below):  **Note**: EADK libraries allow code to be written similar to "standard C" code, which simplifies porting applications to a UEFI environment.

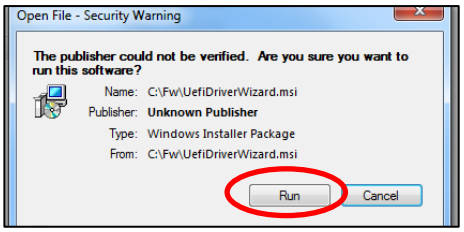
LESSON 8

UEFI DRIVER WIZARD

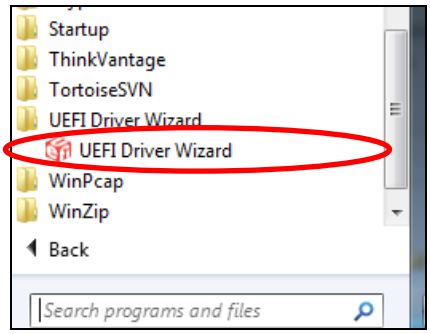
8.1 Creating a UEFI Driver Using the UEFI Driver Wizard

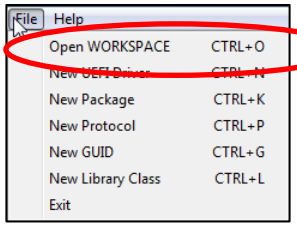
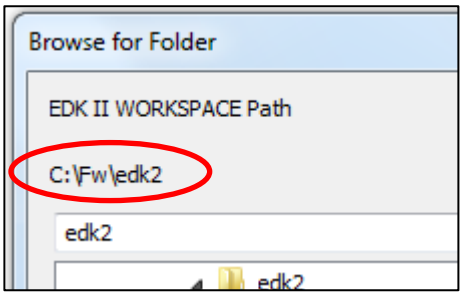
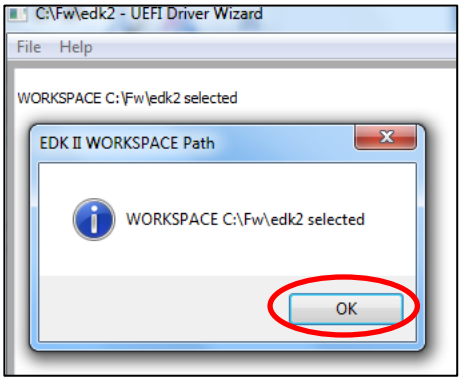
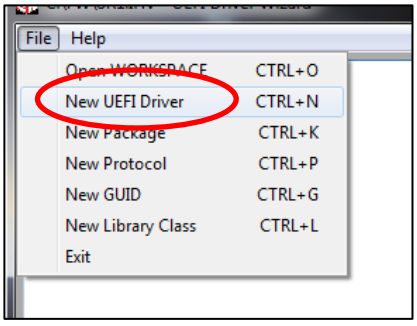
In this lab, you'll create a new UEFI driver using the UEFI driver wizard.

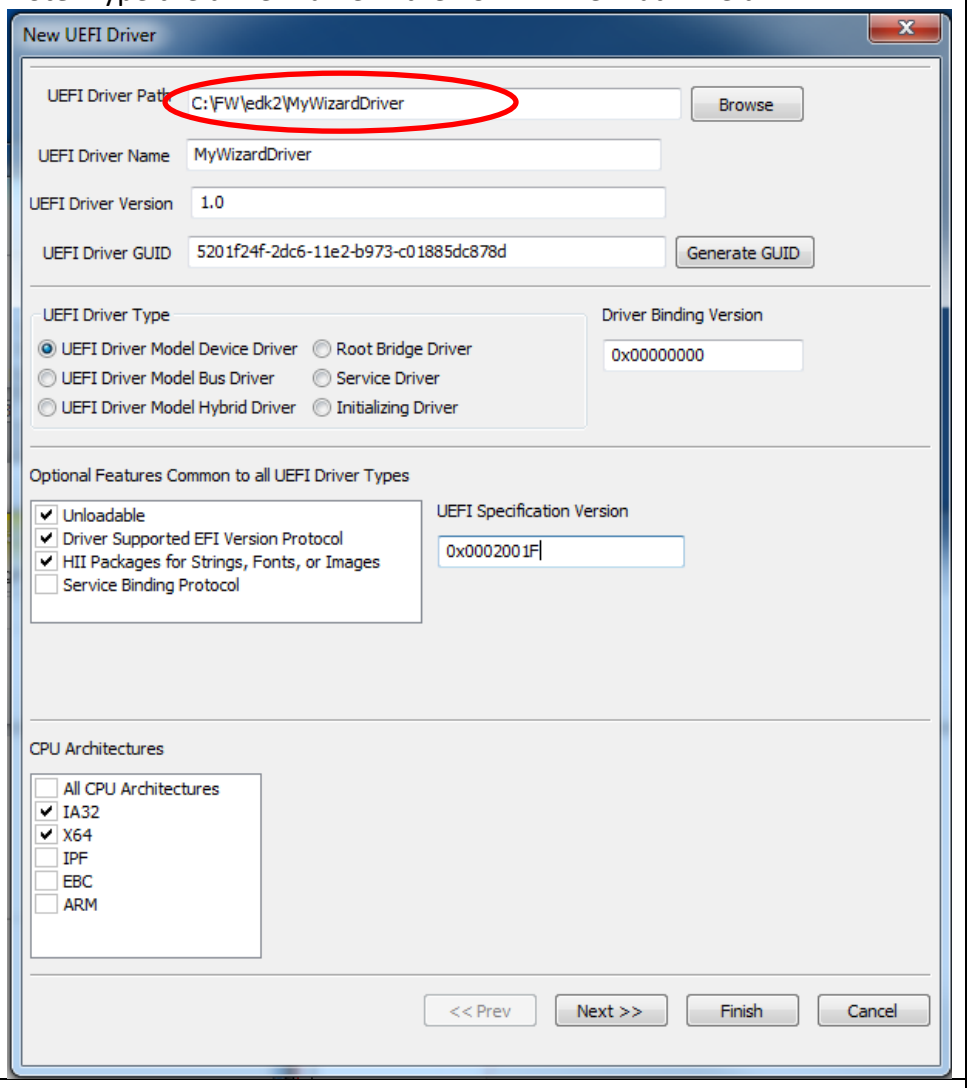
Installing the Driver Wizard

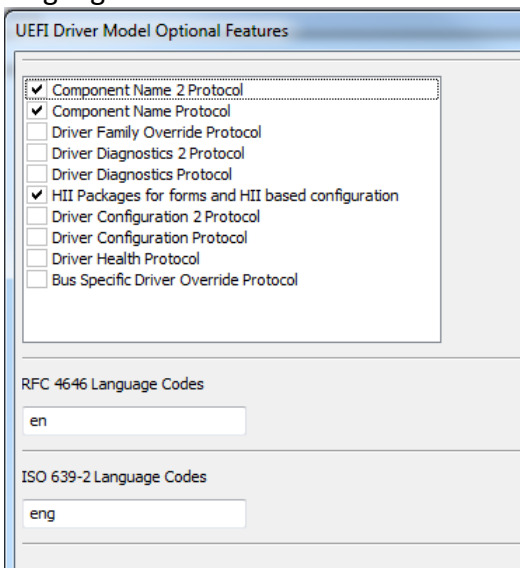
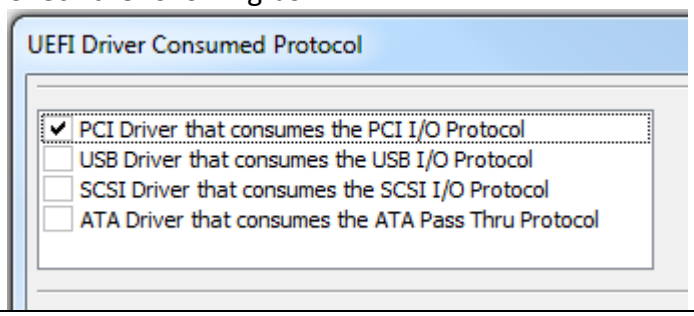
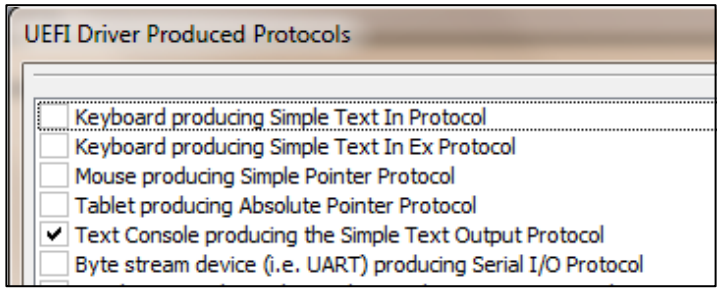
Step	Action
1	**Complete** Section 1.1 above to configure for building with Visual Studio.
2	**Keep** the Visual Studio Command Prompt open
3	**Install** the UEFI Driver Wizard with the following Steps 4 – 10 (Skip to Step 11 if the UEFI Driver Wizard is already installed.
4	**Open** the C:\FW directory in Windows Explorer
5	**Open** C:\Fw\DriverWizard\UefiDriverWizard.msi
6	**Click** Run to acknowledge the security warning 
7	**Click** Next
8	**Click** Next again
9	**Click** Install Note: If a warning message appears, **click** Yes
10	**Click** Finish

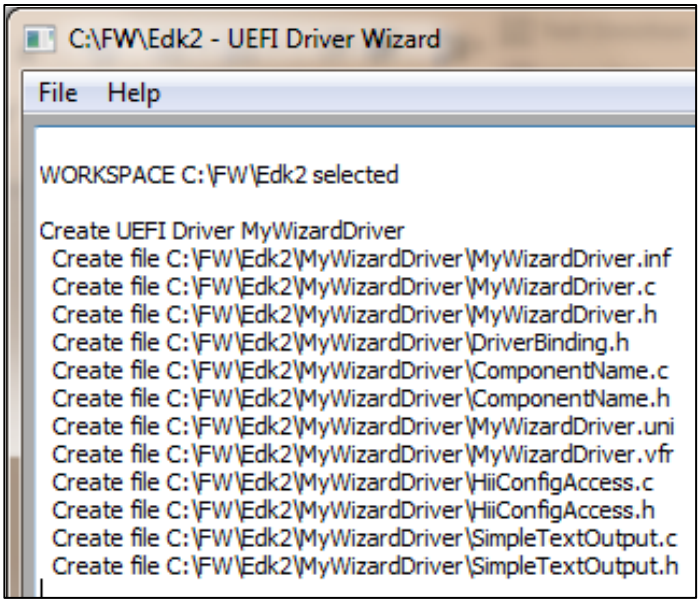
Generating a Template

Step	Action
11	**Open** the UEFI Driver Wizard 
12	**Click** File

Step	Action
13	**Click** Open Workspace 
14	**Click** through the folders to **select** the following workspace path: C:\Fw\edk2 
15	**Click** OK 
16	**Select** New UEFI Driver 

Step	Action
17	**Ensure** all the forms, radio buttons, and boxes are filled in and **selected** ***exactly*** like the image below. ***Note****:* A new, specific driver GUID will populate, so it will be different than this image ***Note***: Type the driver name in the "UEFI Driver Path" field. 
18	**Click** Next

Step	Action
19	**Check** the following boxes: ***Note:*** The language defaults to English, but you can add or delete language codes for Human Interface Infrastructure (HII) here. 
20	**Click** Next
21	Check the following box: 
22	**Click** Next
23	**Check** the following box: 
24	**Click** Finish

Step	Action
25	The driver will create new files in your workspace as follows: 

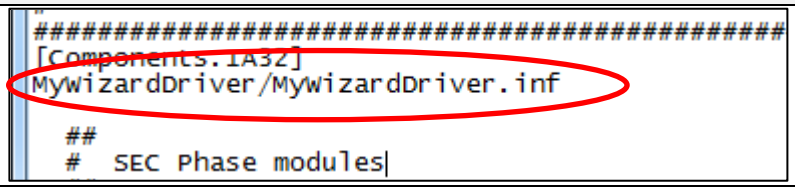
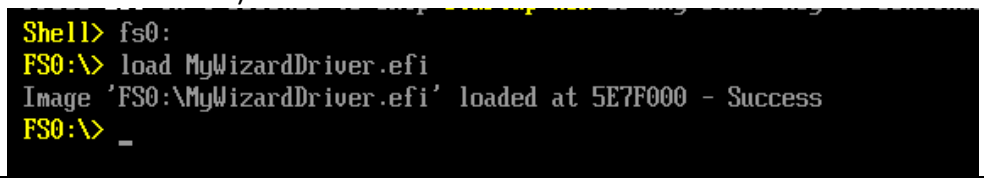
8.2 Building the UEFI Driver

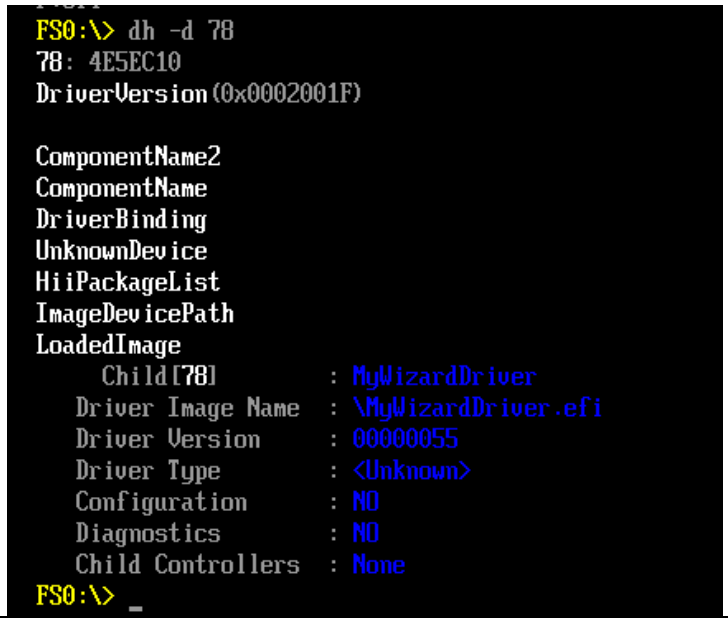
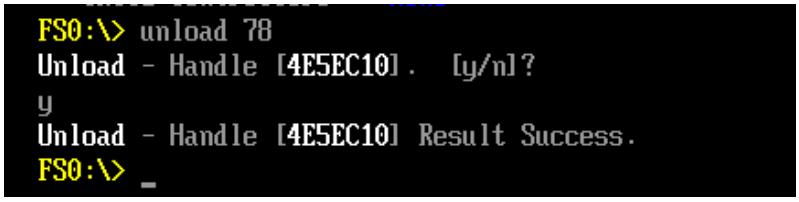
In this lab, you'll build a UEFI Driver. For this lab, you'll include the driver in the NT32 project.

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

Building the UEFI Driver from the Driver Wizard

Step	Action
26	If you haven't performed the EDK II setup (see page 2), do so first
27	**Open** C:\Fw\edk2\Nt32Pkg\Nt32pkg.dsc
28	**Verify** `BUILD_NEW_SHELL` is equal to TRUE
29	**Add** the following to the `[Components.IA32]` section:

	`MyWizardDriver/MyWizardDriver.inf` 
30	**Save** and **close** the file
31	In the Visual Studio Command Prompt, **enter** `cd C:\fw\edk2`
32	**Press** "Enter"
33	**Enter** `edksetup`
34	**Press** "Enter"
35	**Type** `build`
36	**Press** "Enter"
37	**Type** `build run`
38	**Press** "Enter" to start the NT32 Emulation
39	At the Shell 2.0 prompt, **type** `fs0:`
40	**Press** "Enter"
41	**Type** `load MyWizardDriver.efi`
42	**Press** "Enter" to load the driver created by the UEFI Driver Wizard (sample screenshot below): 
43	**Type** `drivers`
44	**Press** "Enter" to verify the UEFI Shell loaded the new driver. The drivers command will display the driver information and a driver handle number ('78' in the example screenshot below). 
45	**Type** `dh -d 78` ***Note***: The value 78 is the driver handle for `MyWizardDriver`. The handle value may change based on your system configuration.

46	**Press** "Enter" to see the information for this driver handle. 
47	**Type** `unload 78`
48	**Press** "Enter" to unload the driver from the UEFI Shell. 
49	**Type** `drivers`
50	**Press** "Enter" to see the results of unloading the driver. ***Note***: Information may still be displayed for `MyWizardDriver`. This will be fixed in Lesson 8.6 Porting Unload () and Stop ( ) Functions.
51	**Type** `Reset` to exit
52	**Press** "Enter" to close the NT32 Emulation.
53	Keep the Visual Studio command prompt (`C:\fw\edk2>`)open for the next lab

8.3 Editing MyWizardDriver/ComponentName.c

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

Step	Action
1	**Open** C:\fw\edk2\MyWizardDriver\ComponentName.c
2	**Change** the string returned by the driver from `MyWizardDriver` (see below) `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 }` `};`
3	**Save** and **close** the file
4	In the Visual Studio Command Prompt, **type** `build`
5	**Press** "Enter" to rebuild the NT32 Emulation
6	**Type** `build run`
7	**Press** "Enter" to start the NT32 Emulation
8	At the Shell 2.0 prompt, **type** `fs0:`
9	**Press** "Enter"
10	**Type** `load MyWizardDriver.efi`
11	Press "Enter"
12	**Type** `Drivers`
13	Press "Enter" to observe the change in the string that the driver returned (see below) `62 0000000A ? N N 0 0 iSCSI Driver Fv(6D99E806-3D38-42C2-A095-5F4300BFD7DC)/FvFile(4579B72D-7EC4-4DD4-8486-083C86B182A7)` `64 0000000A ? N N 0 0 FAT File System Driver Fv(6D99E806-3D38-42C2-A095-5F4300BFD7DC)/FvFile(961578FE-B6B7-44C3-AF35-6BC705CD2B1F)` `78 00000055 ? N N 0 0 UEFI Sample Driver FS0:\MyWizardDriver.efi` `FS0:\>`
14	**Type** `reset`
15	**Press** "Enter" to close the NT32 Emulation.

8.4 Porting the UEFI Driver Support and Start Functions

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

Step	Action
1	**Open** C:\fw\edk2\MyWizardDriver\MyWizardDriver.c
2	Locate `MyWizardDriverDriverBindingSupported()`, the Supported function for this driver and comment out the " `//` " in `//return EFI_UNSUPPORTED;` to achieve the result below: `EFI_STATUS` `EFIAPI` `MyWizardDriverDriverBindingSupported (` `  IN EFI_DRIVER_BINDING_PROTOCOL  *This,` `  IN EFI_HANDLE  ControllerHandle,` `  IN EFI_DEVICE_PATH_PROTOCOL  *RemainingDevicePath OPTIONAL` `  )` `{` `  return EFI_UNSUPPORTED;` `}`
3	**Replace** what you just commented out with the following text (copy and paste). **Note**: This code checks for a specific protocol before returning a status for the supported function (`EFI_SUCCESS` if the protocol GUID exists).

Step	Action
	`` 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; ```
4	**Save** the file, but keep the file **open.**
5	**Open** C:\fw\edk2\MyWizardDriver\MyWizardDriver.h
6	**Add** the following include statement:
	`#include <Protocol/SerialIo.h>`
	`` // // Consumed Protocols // #include <Protocol/PciIo.h> #include <Protocol/SerialIo.h> ```
7	**Save** the file, but keep the file **open.**
8	**Open** C:\fw\edk2\MyWizardDriver\MyWizardDriver.inf
9	Add the following to the end of the `[Protocols]` section:
	`gEfiSerialIoProtocolGuid`

Step	Action
	[Protocols] gEfiDriverBindingProtocolGuid gEfiPciIoProtocolGuid gEfiDriverSupportedEfiVersionProtocolGu gEfiHiiPackageListProtocolGuid gEfiHiiDatabaseProtocolGuid gEfiComponentName2ProtocolGuid gEfiComponentNameProtocolGuid gEfiHiiConfigAccessProtocolGuid gEfiSimpleTextOutProtocolGuid gEfiSerialIoProtocolGuid
10	**Save** and **close** `MyWizardDriver.inf`
11	**Review** the `Libraries` section of `MyWizardDriver.h` // // 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> **Note**: The driver wizard automatically includes library headers based on the form information. The UEFI Driver Wizard already includes headers for common libaries, so these library functions can be used without changing the header file. **Note**: To look up other functions, access the help file: `C:\Fw\Other Files\MdePkg Document With Libraries.chm`
12	**Save** and **close** `MyWizardDriver.h`
13	In MyWizardDriver.c file, **type (copy and paste)** the following text after the `#include "MyWizardDriver.h"` line: `#define DUMMY_SIZE 100*16 // Dummy buffer` `CHAR16 *DummyBufferfromStart = NULL;` #include "MyWizardDriver.h" #define DUMMY_SIZE 100*16 // Dummy buffer CHAR16 *DummyBufferfromStart = NULL;

Step	Action
14	**Locate** `MyWizardDriverDriverBindingStart`, the Start function for this driver and **comment out the** “ `//` “ in `//return EFI_UNSUPPORTED;` for the result below: `EFI_STATUS` `EFIAPI` `MyWizardDriverDriverBindingStart (` `  IN EFI_DRIVER_BINDING_PROTOCOL  *This,` `  IN EFI_HANDLE  ControllerHandle,` `  IN EFI_DEVICE_PATH_PROTOCOL  *RemainingDevicePath OPTIONAL` `  )` `{` `  return EFI_UNSUPPORTED;` `}`
15	**Add** (copy and paste) this code to the function: ***Note***: This code checks for an allocated memory buffer. If the buffer doesn’t exist, memory will be allocated and filled with an initial value (`0x0042`).

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

  //return EFI_UNSUPPORTED;
```

Step	Action

```
EFI_STATUS
EFIAPI
MyWizardDriverDriverBindingStart (
  IN EFI_DRIVER_BINDING_PROTOCOL  *This,
  IN EFI_HANDLE                   ControllerHandle,
  IN EFI_DEVICE_PATH_PROTOCOL     *RemainingDevicePath OPTIONAL
  )
{


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

 //return EFI_UNSUPPORTED;
```

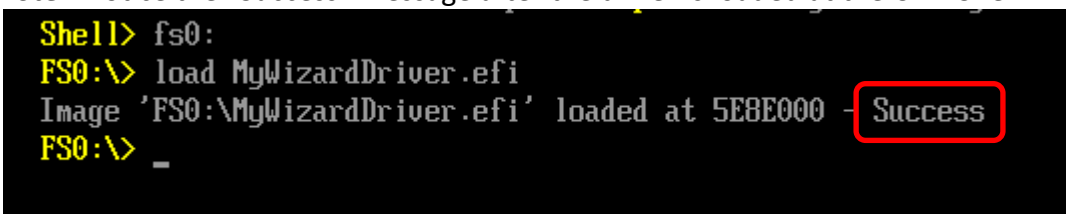
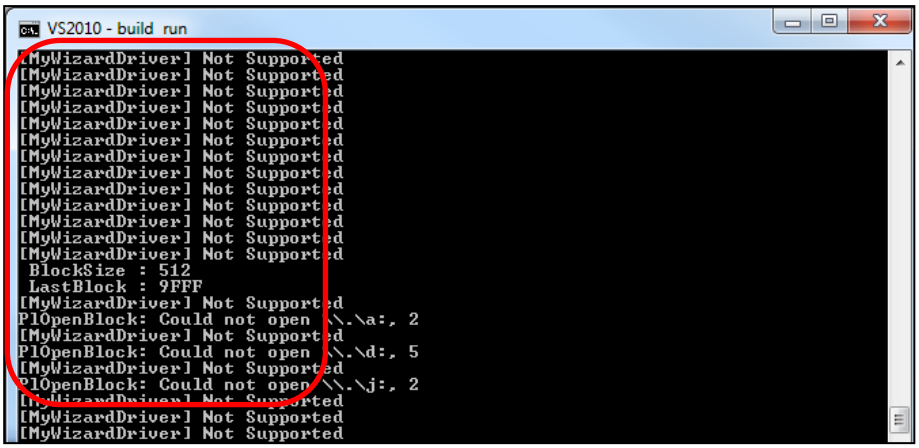
16 Now **add (copy and paste)** the following debug macro into the Supported function:

Note: This string will be displayed on the debug console if the Supported function does not return `EFI_SUCCESS`. The `Status` variable depends on the output of the `OpenProtocol` function.

```
DEBUG ((EFI_D_INFO, "[MyWizardDriver] Not Supported \r\n")
);
```

```
  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
        );
```

Step	Action
17	Go a few lines down and **add (copy and paste)** the following macro to the Supported function, prior to the end of the function: `DEBUG ((EFI_D_INFO,"[MyWizardDriver] Supported SUCCESS\r\n") );` `// 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") );` **Note**: This string will be displayed on the debug console if the Supported function returns `EFI_SUCCESS`.
18	**Locate** the `MyWizardDriverDriverBindingStart` function and **add (copy and paste)** the following debug macro after the memory buffer is filled: `DEBUG ((EFI_D_INFO, "[MyWizardDriver] Buffer 0x%08x\r\n", DummyBufferfromStart));` `if (DummyBufferfromStart == NULL) {` `return EFI_OUT_OF_RESOURCES; // Exit if the buffer isn't there` `}` `SetMem16 (DummyBufferfromStart, (DUMMY_SIZE * sizeof(CHAR16)), 0x0042); // Fill buffer` `DEBUG ((EFI_D_INFO, "[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 (Visual Studio Command Prompt).
19	**Save** MyWizardDriver.c, but **keep the file open** for the next lab
20	If you closed the Visual Studio Command Prompt in the previous lab, or if you haven't performed the EDK II setup (see page 2), do so now. If you have, skip to the next step
21	At the Visual Studio Command Prompt**, type** `build`
22	**Press** "Enter" to rebuild the NT32 project.
23	**Type** `build run`
24	Press "Enter" to start the NT32 Emulation.
25	At the Shell 2.0 prompt, **type** `fs0:`
26	Press "Enter"
27	**Type** `load MyWizardDriver.efi`

Step	Action
28	Press “Enter” ***Note:*** Notice the “Success” message after the driver is loaded at the UEFI Shell.  **Note**: Check the debug console output (Visual Studio Command Prompt).  **Note**: Debug messages indicate the driver did not return EFI_SUCCESS from the “Supported” function. This issue will be fixed in the next lab 8.5 Returning EFI_SUCCESS from the Supported Function
29	**Type** `reset`
30	**Press** “Enter” to close the NT32 Emulation.
31	**Keep** the Visual Studio Command prompt open for the next lab

8.5 Returning EFI_SUCCESS from the Supported Function

In this lab, you will fix the driver's "Supported" function. On systems without a serial port, the code from previous lab will not work since the Serial Protocol GUID does not exist. This lab demonstrates another mechanism (also known as "a trick") to return from the "Supported" function.

Step	Action
1	**Create** a new file in your editor: MyWizardDriverNVDataStruc.h
2	**Add (copy and paste)** the following into the MyWizardDriverNVDataStruc.h file:

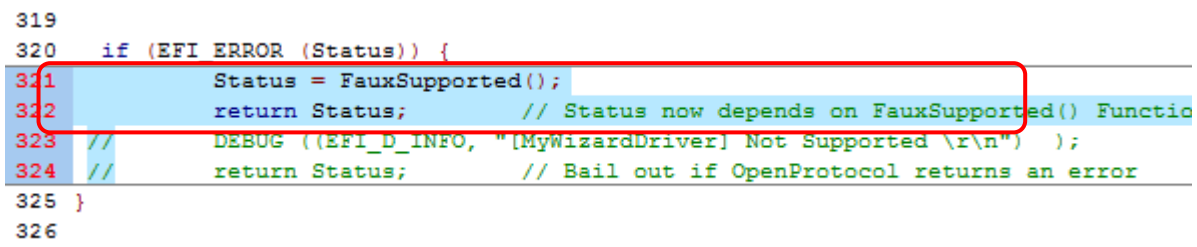
```
#ifndef _MYWIZARDDRIVERNVDATASTRUC_H_
#define _MYWIZARDDRIVERNVDATASTRUC_H_
#include <Guid/HiiPlatformSetupFormset.h>
#include <Guid/HiiFormMapMethodGuid.h>

#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
```

Note: In the following labs you'll add a UEFI variable. In order to set, retrieve, and use the UEFI variable, it requires the GUID reference that you just added.

Note: For this lab, you were provided a GUID file. You can also generate a GUID through the UEFI Driver Wizard or Guidgenerator.com. But for the purposes of the lab, use the one above.

3	**Save** the file to C:\FW\edk2\MyWizardDriverNVDataStruc.h and **close** it
4	**Open** C:\FW\edk2\MyWizardDriver\MyWizardDriver.c
5	**Add (copy and paste)** the following text after the `#include "MyWizardDriver.h"` statement:

Step	Action
	`EFI_GUID mMyWizardDriverVarGuid = MYWIZARDDRIVER_VAR_GUID;` `CHAR16 mVariableName[] = L"MWD_NVData";` `MYWIZARDDRIVER_CONFIGURATION mMyWizDrv_Conf_buffer;` `MYWIZARDDRIVER_CONFIGURATION *mMyWizDrv_Conf =` `&mMyWizDrv_Conf_buffer; //use the pointer`
	12 #include "MyWizardDriver.h" 13 14 EFI_GUID mMyWizardDriverVarGuid = MYWIZARDDRIVER_VAR_GUID; 15 16 CHAR16 mVariableName[] = L"MWD_NVData"; 17 MYWIZARDDRIVER_CONFIGURATION mMyWizDrv_Conf_buffer; 18 MYWIZARDDRIVER_CONFIGURATION *mMyWizDrv_Conf = &mMyWizDrv_Conf_buffer; // 19 20 21 // *** LAB *** 22 #define DUMMY_SIZE 100*16 // Dummy buf 23 CHAR16 *DummyBufferfromStart = NULL;
6	**Locate** the `MyWizardDriverDriverBindingSupported` function then **comment out** the DEBUG macro statement and `return` statement as shown below: `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;`
7	**Add (copy and paste)** the following text above the commented out debug macro `Status = FauxSupported();` `return Status; // Status now depends on FauxSupported Function`

Step	Action
	 **Note**: If the Supported function tries to return an error, this new function will be checked and may change the return code based on NVRAM contents.
8	**Add (copy and paste)** the following text below the function definition for `MyWizardDriverDriverEntryPoint` and before the function `MyWizardDriverDriverBindingSupported:`

```
EFI_STATUS
EFIAPI
FauxSupported()
{
       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] Supported
SUCCESS with Faux Supported by NVRam Var\r\n"));
                  return EFI_SUCCESS;
             }
      }
      // already defined once so this time through unsupported
      return EFI_UNSUPPORTED;
}
```

9	**Save** and **close** `MyWizardDriver.c`

Step	Action
10	**Open** C:\FW\edk2\MyWizardDriver\ MyWizardDriver.h
11	**Edit `MyWizardDriver.h`** and add (copy and paste) the following to the list of Libraries:
	`// LAB FAUX` `#include <Library/UefiRuntimeServicesTableLib.h>` `// LAB FAUX`
	`#include <Library/DebugLib.h>` `//LAB FAUX` `#include <Library/UefiRuntimeServicesTableLib.h>` `// END LAB`
12	Add include for the new .h file for the new .h file after `#include <Protocol/SimpleTextOut.h>`
	`#include "MyWizardDriverNVDataStruc.h"`
	`// Produced Protocols` `//` `#include <Protocol/SimpleTextOut.h>` `#include "MyWizardDriverNVDataStruc.h"` `//`
13	**Save** and **close** MyWizardDriver.h
14	At the Visual Studio Command Prompt**, type** `build`
15	**Press** "Enter" to rebuild the NT32 project.
16	**Type** `build run`
17	**Press** "Enter" to start the NT32 Emulation.
18	At the Shell 2.0 prompt, **type** `fs0:`
19	**Press** "Enter"
20	**Type** `load MyWizardDriver.efi`
21	**Type** the `mem` command to verify that the driver loaded properly by checking the memory buffer.
22	**Press** "Enter"
23	At the Shell 2.0 prompt**, type** the `mem` command followed by the buffer address to display the buffer contents. For example, if you were using the following buffer, you would enter `mem 0x0509B010` `[MyWizardDriver] Supported SUCCESS with Faux Supported by NVRam Var` `[MyWizardDriver] Buffer 0x0509B010` ***Note:*** If the buffer address starts with a letter, you must add a zero before the letter.

Step	Action
24	**Press** "Enter" to display the memory buffer. If the driver Start function was excuted properly, **mem** will show the 0x0042 value used to fill the buffer. `Shell> fs0:` `FS0:\> load MyWizardDriver.efi` `Image 'FS0:\MyWizardDriver.efi' loaded at 60BE000 - Success` `FS0:\> mem 0509b010` `Memory Address 000000000509B010 200 Bytes` `0509B010: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00 *B.B.B.B.B.B.B.B.*` `0509B020: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00 *B.B.B.B.B.B.B.B.*` `0509B030: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00 *B.B.B.B.B.B.B.B.*` `0509B040: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00 *B.B.B.B.B.B.B.B.*` `0509B050: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00 *B.B.B.B.B.B.B.B.*` `0509B060: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00 *B.B.B.B.B.B.B.B.*` `0509B070: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00 *B.B.B.B.B.B.B.B.*` `0509B080: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00 *B.B.B.B.B.B.B.B.*` `0509B090: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00 *B.B.B.B.B.B.B.B.*`
25	At the Shell 2.0 prompt, **type** `reset`
26	**Press** "Enter" to close the NT32 Emulation.
27	**Keep** the Visual Studio Command prompt open for the next lab

8.6 Porting `Unload()` and `Stop()` Functions

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.

Step	Action
1	**Open** C:\fw\edk2\MyWizardDriver\MyWizardDriver.c
2	**Locate** the unload function `MyWizardDriverUnload` then **add (copy and paste)** the following text shown below before the return EFI_SUCCESS:

```
  if (DummyBufferfromStart != NULL) {
     FreePool(DummyBufferfromStart);
     DEBUG ((EFI_D_INFO,"[MyWizardDriver] Unload, clear
buffer\r\n"));
     }
  DEBUG ((EFI_D_INFO,"[MyWizardDriver] Unload success\r\n"));
```

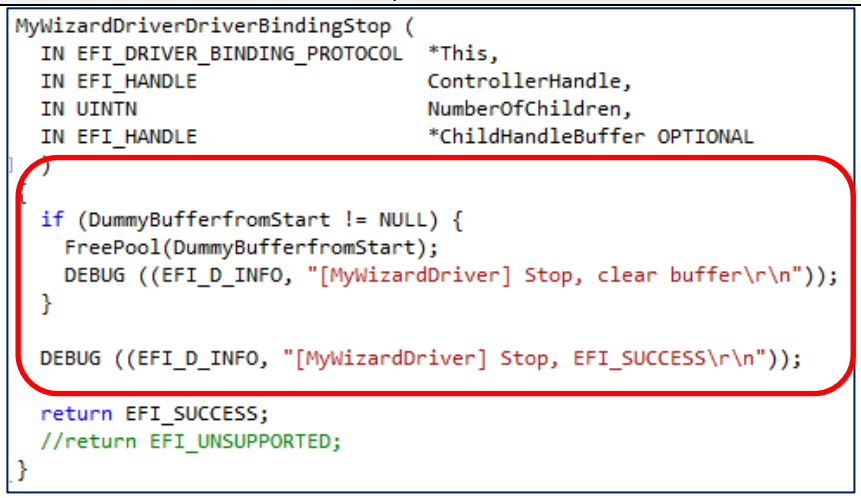
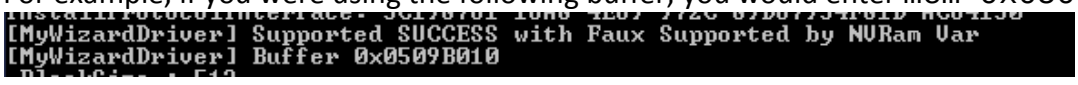
```
119
120   //
121   // Do any additional cleanup that is required for this driver
122   //
123   if (DummyBufferfromStart != NULL) {
124         FreePool(DummyBufferfromStart);
125         DEBUG ((EFI_D_INFO, "[MyWizardDriver] Unload, clear buffer\r\n"));
126   }
127   DEBUG ((EFI_D_INFO, "[MyWizardDriver] Unload success\r\n"));
128
129   return EFI_SUCCESS;
130 }
```

Note: The code will deallocate the buffer using the **`FreePool`** library function.

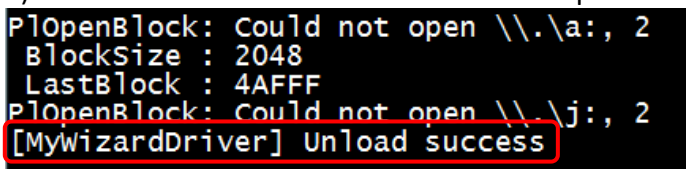
3	**Locate** the stop function MyWizardDriverDriverBindingStop and **comment out** the `return EFI_UNSUPPORTED;` statement.

```
EFI_STATUS
EFIAPI
MyWizardDriverDriverBindingStop (
  IN EFI_DRIVER_BINDING_PROTOCOL  *This,
  IN EFI_HANDLE                   ControllerHandle,
  IN UINTN                        NumberOfChildren,
  IN EFI_HANDLE                   *ChildHandleBuffer OPTIONAL
  )
{
  return EFI_UNSUPPORTED;
}
```

4	**Add (copy and paste)** the following text to body of the function (shown below):

Step	Action
	`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;`
	 ***Note***: This code will deallocate the buffer using the **`FreePool`** library function. This a duplicate of the check performed in the unload function, in case the stop function was executed prior to unload.
5	**Save** and **close** the file.
6	At the Visual Studio Command Prompt**, type** `build`
7	**Press** "Enter" to rebuild the project.
8	**Type** `build run`
9	**Press** "Enter" to start the NT32 Emulation.
10	At the Shell 2.0 prompt, **type** `fs0:`
11	**Press** "Enter"
12	**Type** `load MyWizardDriver.efi`
13	**Press** "Enter" to load the driver.
14	At the Shell 2.0 prompt**, type** the `mem` command followed by the buffer address to display the buffer contents. For example, if you were using the following buffer, you would enter `mem 0x0509B010`  ***Note:*** If the buffer address starts with a letter, you must add a zero before the letter.

Step	Action
15	**Press** "Enter" to display the memory buffer. If the driver Start function was excuted properly, **mem** will show the 0x0042 value used to fill the buffer. `Shell> fs0:` `FS0:\> load MyWizardDriver.efi` `Image 'FS0:\MyWizardDriver.efi' loaded at 60BE000 - Success` `FS0:\> mem 0509b010` `Memory Address 000000000509B010 200 Bytes` `0509B010: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00 *B.B.B.B.B.B.B.B.*` `0509B020: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00 *B.B.B.B.B.B.B.B.*` `0509B030: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00 *B.B.B.B.B.B.B.B.*` `0509B040: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00 *B.B.B.B.B.B.B.B.*` `0509B050: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00 *B.B.B.B.B.B.B.B.*` `0509B060: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00 *B.B.B.B.B.B.B.B.*` `0509B070: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00 *B.B.B.B.B.B.B.B.*` `0509B080: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00 *B.B.B.B.B.B.B.B.*` `0509B090: 42 00 42 00 42 00 42 00-42 00 42 00 42 00 42 00 *B.B.B.B.B.B.B.B.*`
16	Now, test the "Unload" and "Stop" functions. At the Shell 2.0 prompt, **type** `drivers`
17	**Press** "Enter" to run the drivers command and retrieve the handle number associated with `MyWizardDriver` `64 0000000A ? N N 0 0 FAT File System Driver Fv(6D99E806-3D38-42C2-A095-5F4300BFD7DC)/FvFile(961578FE-B6B7-44C3-AF35-6BC705CD2B1F)` `78 00000055 ? N N 0 0 UEFI Sample Driver FS0:\\MyWizardDriver.efi` `FS0:\> _` ***Note***: In this example, the driver handle is **78**.
18	At the Shell 2.0 prompt, use the `unload` command to unload the driver by **typing** `unload 78` ***Note:*** The handle number may be different for your configuration
19	**Press** "Enter" then **press** "y" to unload the driver. `FS0:\> unload 78` `Unload - Handle [509EC10]. [y/n]?` `y` `Unload - Handle [509EC10] Result Success.` `FS0:\> _`

Step	Action
20	Confirming that the driver was properly unloaded requires two steps: 1) **Check** the Visual Studio Command Prompt for the DEBUG message.  2) **Re-run** the drivers command to verify the driver handle is not present. 
21	At the Shell 2.0 prompt, **type** `reset`
22	**Press** “Enter” to close the NT32 Emulation.

LESSON 9

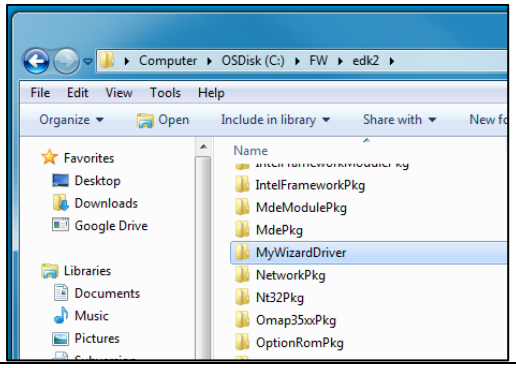
UEFI DRIVER WIZARD – ADDING HII

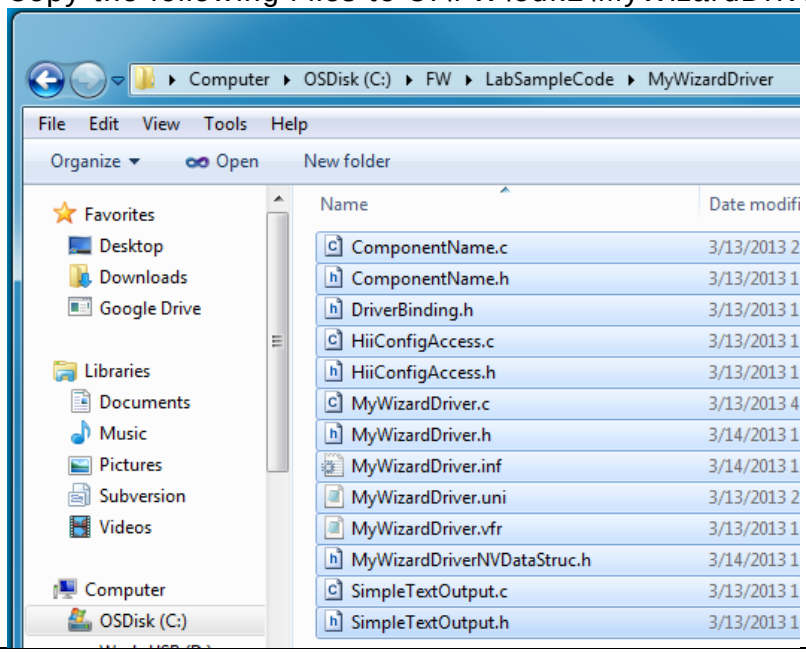
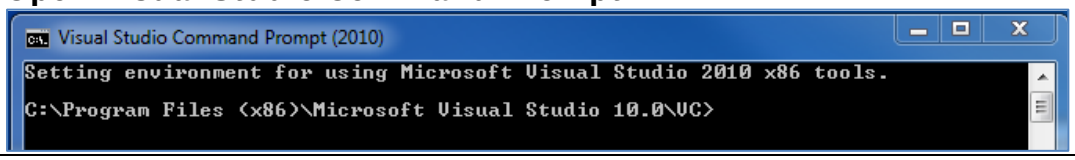
9.1 Adding Strings and Forms to Setup HII for User Configuration

In this lab, you'll learn how to use HII to add strings and forms to a firmware setup menu for user configuration. Once you've complete this lab, your end result will match Figure 1.

Figure 1 My Wizard Driver menu with strings and forms

9.1.1 Setup for Lab adding HII

Step	Action
1	Complete Section 1.1 above to configure for building with Visual Studio.
2	Start with LAB 8.6 solution and **create** a folder called **MyWizardDriver** in the C:\fw\edk2 workspace 
3	Now**, locate** and **open**: C:\FW\LabSampleCode\MyWizardDriver

Step	Action
4	Copy the following Files to C:\FW\edk2\MyWizardDriver 
5	**Open Visual Studio Command Prompt** 
6	**Type** CD C:\fw\edk2
7	**Type** Edksetup (This is only needed if starting out with a new Visual Studio Command Prompt Window)

9.1.2 Edit Driver for adding HII

Step	Action
1	**Open** C:\fw\edk2\MyWizardDriver
2	**Open** the following files for updating: 1) MyWizardDriverNVDataStruc.h 2) MyWizardDriver.vfr 3) MyWizardDriver.uni 4) MyWizardDriver.h 5) MyWizardDriver.c 6) MyWizardDriver.inf
3	**Update** the MyWizardDriverNVDataStruc.h file by copying and pasting the following GUID as shown below: This GUID is used to communicate to the HII Database and Browser Engine

```
#define MYWIZARDDRIVER_FORMSET_GUID \
  { \
0x5481db09, 0xe5f7, 0x4158, 0xa5, 0xc5, 0x2d, 0xbe, 0xa4, 0x95,
0x34, 0xff \
}
```

```
6
7  #define MYWIZARDDRIVER_VAR_GUID \
8    { \
9          0x363729f9, 0x35fc, 0x40a6, 0xaf, 0xc8, 0xe8, 0xf5, 0x49, 0x11, 0xf1, 0xd6 \
10   }
11
12 #define MYWIZARDDRIVER_FORMSET_GUID \
13   { \
14         0x5481db09, 0xe5f7, 0x4158, 0xa5, 0xc5, 0x2d, 0xbe, 0xa4, 0x95, 0x34, 0xff \
15  }
16
17
18 #pragma pack(1)
19 typedef struct {
20
21     UINT16  MyWizardDriverStringData[20];
22     UINT8   MyWizardDriverHexData;
23     UINT8   MyWizardDriverBaseAddress;
```

4	**Save** MyWizardDriverNVDataStruc.h
5	**Update** the **MyWizardDriver.vfr** file. **Delete** its contents and **replace** it with the following by copying and pasting: You're adding a reference to the GUID and to the NVRAM storage where the configuration will be saved. In fact, you're replacing most of the original .vfr.

```
#include "MyWizardDriverNVDataStruc.h"
formset
  guid     = MYWIZARDDRIVER_FORMSET_GUID,
  title    = STRING_TOKEN(STR_SAMPLE_FORM_SET_TITLE),
  help     = STRING_TOKEN(STR_SAMPLE_FORM_SET_HELP),
  classguid = EFI_HII_PLATFORM_SETUP_FORMSET_GUID,

  //
  // Define a Buffer Storage (EFI_IFR_VARSTORE)
  //
    varstore MYWIZARDDRIVER_CONFIGURATION,  // This is the data
structure type
    //varid = CONFIGURATION_VARSTORE_ID,      // Optional VarStore
ID
    name  = MWD_IfrNVData,                  // Define referenced
name in vfr
    guid  = MYWIZARDDRIVER_FORMSET_GUID;    // GUID of this buffer
storage
```

6

Continue **adding** the remaining code to MyWizardDriver.vfr.

This is a Enable/ Disable question for the setup menu in the form of a Check box.

```
  form formid = 1, title = STRING_TOKEN(STR_SAMPLE_FORM1_TITLE);
    subtitle text = STRING_TOKEN(STR_SUBTITLE_TEXT);

    subtitle text = STRING_TOKEN(STR_SUBTITLE_TEXT2);

  //
  // Define a checkbox to enable / disable the device
  //
      checkbox varid   = MWD_IfrNVData.MyWizardDriverChooseToEnable,
                 prompt   = STRING_TOKEN(STR_CHECK_BOX_PROMPT),
                 help     = STRING_TOKEN(STR_CHECK_BOX_HELP),
                 //
                 // CHECKBOX_DEFAULT indicate this checkbox is
marked with
           //   EFI_IFR_CHECKBOX_DEFAULT
                 //
                 flags    = CHECKBOX_DEFAULT ,
                 key      = 0,
                 default  = 1,
     endcheckbox;

   endform;


endformset;
```

7

Save MyWizardDriver.vfr

8

Now onto the MyWizardDriver.uni file. You'll add new strings to support the forms. **Delete** the file's content and **replace** it with the following by copying and pasting:

```
#langdef en "English"

#string STR_SAMPLE_FORM_SET_TITLE      #language en  "My Wizard Driver Sample
Formset"
#string STR_SAMPLE_FORM_SET_HELP       #language en  "Help for Sample Formset"
#string STR_SAMPLE_FORM1_TITLE         #language en  "My Wizard Driver"

#string STR_SUBTITLE_TEXT              #language en "My Wizard Driver
Configuration"
#string STR_SUBTITLE_TEXT2             #language en "Device XYZ Configuration"
#string STR_CHECK_BOX_PROMPT           #language en "Enable My XYZ Device"


#string STR_CHECK_BOX_HELP             #language en "This is the help message
for the enable My XYZ device. Check this box to enable this device."
```

9

Save MyWizardDriver.uni

10

Now update the MyWizardDriver.h file. **Add** the following HII libraries starting at approximately line 41 (as shown below) by copying and pasting:

By adding this code, now your driver will be consuming the HII Protocols and producing the CONFIG ACCESS PROTOCOL:

```
// Added for HII
#include <Protocol/HiiConfigRouting.h>
#include <Protocol/FormBrowser2.h>
#include <Protocol/HiiString.h>
#include <Library/DevicePathLib.h>
```

```
41
42  // Added for HII
43  #include <Protocol/HiiConfigRouting.h>
44  #include <Protocol/FormBrowser2.h>
45  #include <Protocol/HiiString.h>
46  #include <Library/DevicePathLib.h>
47
48  //
49  // Consumed Protocols
```

11

To add a data structure for HII routing and access, **add** the following code at approximately line 75 by copying and pasting after the "extern" statements:

11

```
#define MYWIZARDDRIVER_DEV_SIGNATURE SIGNATURE_32 ('m', 'w', 'd',
'r')

// Need a Data structure for HII routing and accessing
typedef struct {
  UINT32                            Signature;

  EFI_HANDLE                        Handle;
  MYWIZARDDRIVER_CONFIGURATION    Configuration;

  EFI_HANDLE                        DriverHandle[2];
  EFI_HII_HANDLE                    HiiHandle[2];
  //
  // Consumed protocol
  //
  EFI_HII_DATABASE_PROTOCOL         *HiiDatabase;
  EFI_HII_STRING_PROTOCOL           *HiiString;
  EFI_HII_CONFIG_ROUTING_PROTOCOL   *HiiConfigRouting;
  EFI_FORM_BROWSER2_PROTOCOL        *FormBrowser2;

  //
  // Produced protocol
  //
  EFI_HII_CONFIG_ACCESS_PROTOCOL    ConfigAccess;

} MYWIZARDDRIVER_DEV;

#define MYWIZARDDRIVER_DEV_FROM_THIS(a)  CR (a, MYWIZARDDRIVER_DEV,
ConfigAccess, MYWIZARDDRIVER_DEV_SIGNATURE)

#pragma pack(1)
///
/// HII specific Vendor Device Path definition.
///
typedef struct {
  VENDOR_DEVICE_PATH              VendorDevicePath;
  EFI_DEVICE_PATH_PROTOCOL        End;
} HII_VENDOR_DEVICE_PATH;


#pragma pack()
```

11

```
73  extern EFI_HII_CONFIG_ACCESS_PROTOCOL  gMyWizardDriverHiiConfigAccess;
74
75  #define MYWIZARDDRIVER_DEV_SIGNATURE SIGNATURE_32 ('m', 'w', 'd', 'r')
76
77  // Need a Data structure for HII routing and accessing
78  typedef struct {
79    UINT32                          Signature;
80
81    EFI_HANDLE                      Handle;
82    MYWIZARDDRIVER_CONFIGURATION    Configuration;
83
84    EFI_HANDLE                      DriverHandle[2];
85    EFI_HII_HANDLE                  HiiHandle[2];
86    //
87    // Consumed protocol
88    //
89    EFI_HII_DATABASE_PROTOCOL       *HiiDatabase;
90    EFI_HII_STRING_PROTOCOL         *HiiString;
91    EFI_HII_CONFIG_ROUTING_PROTOCOL *HiiConfigRouting;
92    EFI_FORM_BROWSER2_PROTOCOL      *FormBrowser2;
93
94    //
95    // Produced protocol
96    //
97    EFI_HII_CONFIG_ACCESS_PROTOCOL  ConfigAccess;
98
99  } MYWIZARDDRIVER_DEV;
100
101 #define MYWIZARDDRIVER_DEV_FROM_THIS(a)  CR (a, MYWIZARDDRIVER_DEV, Config
102
103 #pragma pack(1)
104 ///
105 /// HII specific Vendor Device Path definition.
106 ///
107 typedef struct {
108   VENDOR_DEVICE_PATH              VendorDevicePath;
109   EFI_DEVICE_PATH_PROTOCOL        End;
110 } HII_VENDOR_DEVICE_PATH;
111
112 #pragma pack()
113 //
114 // Include files with function prototypes
```

12

Save MyWizardDriver.h

13

Now onto the MyWizardDriver.c file.

Add local definitions for the form GUID, variable name, and device path for HII at approximately line 13 after the `#include "MyWizardDriver.h"` by coping and pasting the following code.

In this step, you declare a local (to the module "m") variable for the GUID we declared; the NVRAM variable name; driver handles; our configuration data; and the device path support.

13

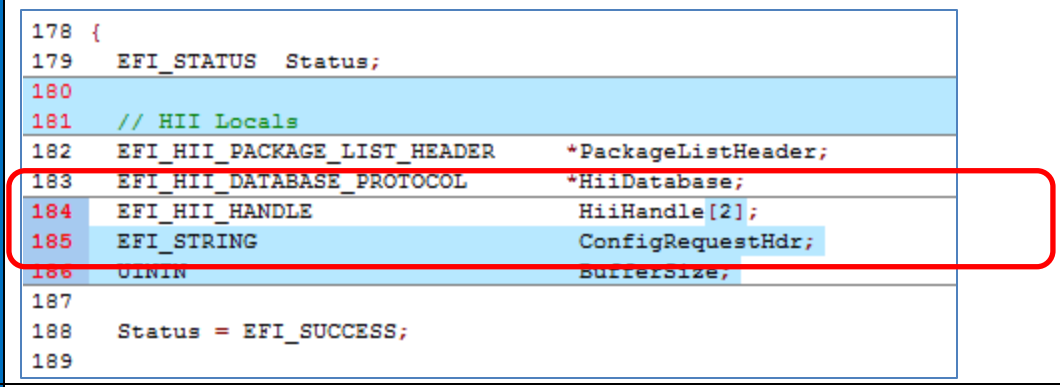
```
//HII support
EFI_GUID   mMyWizardDriverFormSetGuid = MYWIZARDDRIVER_FORMSET_GUID;

CHAR16     mIfrVariableName[] = L"MWD_IfrNVData";
EFI_HANDLE                      mDriverHandle[2] = {NULL, NULL};
MYWIZARDDRIVER_DEV                 *PrivateData = NULL;

// HII support for Device Path
HII_VENDOR_DEVICE_PATH  mHiiVendorDevicePath = {
  {
    {
      HARDWARE_DEVICE_PATH,
      HW_VENDOR_DP,
      {
        (UINT8) (sizeof (VENDOR_DEVICE_PATH)),
        (UINT8) ((sizeof (VENDOR_DEVICE_PATH)) >> 8)
      }
    },
    MYWIZARDDRIVER_FORMSET_GUID
  },
  {
    END_DEVICE_PATH_TYPE,
    END_ENTIRE_DEVICE_PATH_SUBTYPE,
    {
      (UINT8) (END_DEVICE_PATH_LENGTH),
      (UINT8) ((END_DEVICE_PATH_LENGTH) >> 8)
    }
  }
};
```

14

Locate `EFI_STATUS` within the function `MyWizardDriverDriverEntryPoint` in the MyWizardDriver.c file (approx. Line 184) and **add** HII local definitions by copying and pasting (as shown below):

14

```
  // HII Locals
  EFI_HII_PACKAGE_LIST_HEADER     *PackageListHeader;
  EFI_HII_DATABASE_PROTOCOL       *HiiDatabase;
  EFI_HII_HANDLE                   HiiHandle[2];
  EFI_STRING                       ConfigRequestHdr;
  UINTN                            BufferSize;
```

14	
15	**Locate** the **ASSERT_EFI_ERROR (Status);** statement and the line: **// Retrieve HII Package List Header on ImageHandle** (approximately line 202). Now, **add** the following code to install the configuration access protocol (produced) by copying and pasting (as shown below) before the line:**// Retrieve HII Package List Header on ImageHandle**

```
178 {
179   EFI_STATUS  Status;
180
181   // HII Locals
182   EFI_HII_PACKAGE_LIST_HEADER      *PackageListHeader;
183   EFI_HII_DATABASE_PROTOCOL        *HiiDatabase;
184   EFI_HII_HANDLE                    HiiHandle[2];
185   EFI_STRING                        ConfigRequestHdr;
186   UINTN                             BufferSize;
187
188   Status = EFI_SUCCESS;
189
```

15

```
//
//Now do HII Stuff
// Initialize the local variables.
  ConfigRequestHdr = NULL;

// Initialize driver private data
  PrivateData = AllocateZeroPool (sizeof (MYWIZARDDRIVER_DEV));
  if (PrivateData == NULL) {
    return EFI_OUT_OF_RESOURCES;
  }

  PrivateData->Signature = MYWIZARDDRIVER_DEV_SIGNATURE;

  PrivateData->ConfigAccess.ExtractConfig =
MyWizardDriverHiiConfigAccessExtractConfig;
  PrivateData->ConfigAccess.RouteConfig =
MyWizardDriverHiiConfigAccessRouteConfig;
  PrivateData->ConfigAccess.Callback =
MyWizardDriverHiiConfigAccessCallback;

  //
  // Publish sample Fromset and config access
  //
  Status = gBS->InstallMultipleProtocolInterfaces (
                  &mDriverHandle[0],
                  &gEfiDevicePathProtocolGuid,
                  &mHiiVendorDevicePath,
                  &gEfiHiiConfigAccessProtocolGuid,
                  &PrivateData->ConfigAccess,
                  NULL
                  );
  ASSERT_EFI_ERROR (Status);

  PrivateData->DriverHandle[0] = mDriverHandle[0];
```

15

```
201   ASSERT_EFI_ERROR (Status);
202
203   //
204   //Now do HII Stuff
205   //
206
207   // Initialize the local variables.
208   ConfigRequestHdr = NULL;
209   //
210   // Initialize driver private data
211   //
212   PrivateData = AllocateZeroPool (sizeof (MYWIZARDDRIVER_DEV));
213   if (PrivateData == NULL) {
214     return EFI_OUT_OF_RESOURCES;
215   }
216
217   PrivateData->Signature = MYWIZARDDRIVER_DEV_SIGNATURE;
218
219   PrivateData->ConfigAccess.ExtractConfig = MyWizardDriverHiiConfigAccess
220   PrivateData->ConfigAccess.RouteConfig = MyWizardDriverHiiConfigAccessRc
221   PrivateData->ConfigAccess.Callback = MyWizardDriverHiiConfigAccessCallb
222
223
224   //
225   // Publish sample Fromset and config access
226   //
227   Status = gBS->InstallMultipleProtocolInterfaces (
228                  &mDriverHandle[0],
229                  &gEfiDevicePathProtocolGuid,
230                  &mHiiVendorDevicePath,
231                  &gEfiHiiConfigAccessProtocolGuid,
232                  &PrivateData->ConfigAccess,
233                  NULL
234                  );
235   ASSERT_EFI_ERROR (Status);
236
237   PrivateData->DriverHandle[0] = mDriverHandle[0];
238   //
239   // Retrieve HII Package List Header on ImageHandle
240   //
241   Status = gBS->OpenProtocol (
```

16

Next, **add** code to register a list of HII packages in the HII Database with the HII device path. This requires you to **replace** existing code (see below) by copying and pasting the new code at approx. line 265.

Find: `// Register list of HII packages in the HII Database` and replace

```
                                NULL,
                                &HiiHandle
```

The HII Browser will need to find your HII Package and it does this when the call is made to NewPackageList with the device path of your driver's HII packages. The mDriverHandle is your Driver's Device path. Use this in the call to NewPackageList instead of the NULL parameter used before.

16

Old Code

```
190                     );
191     if (!EFI_ERROR (Status)) {
192       //
193       // Register list of HII packages in the HII Database
194       //
195       Status = HiiDatabase->NewPackageList (
196                                 HiiDatabase,
197                                 PackageListHeader,
198                                 NULL,
199                                 &HiiHandle
200                                 );
201       ASSERT_EFI_ERROR (Status);
202     }
203   }
204   Status = EFI_SUCCESS;
205
206   //
```

16

```
mDriverHandle[0],
&HiiHandle[0]
```

16

New Code

```
257                     );
258     if (!EFI_ERROR (Status)) {
259       //
260       // Register list of HII packages in the HII Database
261       //
262       Status = HiiDatabase->NewPackageList (
263                                 HiiDatabase,
264                                 PackageListHeader,
265                                 mDriverHandle[0],
266                                 &HiiHandle[0]
267                                 );
268       ASSERT_EFI_ERROR (Status);
269     }
270   }
271   Status = EFI_SUCCESS;
```

17

Next, you'll **add** code to initialize the My Wizard Driver NVRAM variable by copying and pasting the following code **before** the `// Install Driver Supported EFI Version Protocol onto ImageHandle` comment (as shown below at approximately line 273):

17

```
  PrivateData->HiiHandle[0] = HiiHandle[0];

  BufferSize = sizeof (MYWIZARDDRIVER_CONFIGURATION);

  // IF driver is not part of the Platform then need to get/set
defaults for the NVRAM configuration that the driver will use.
  Status = gRT->GetVariable (
            mIfrVariableName,
            &mMyWizardDriverFormSetGuid,
            NULL,
            &BufferSize,
            &PrivateData->Configuration
            );
  if (EFI_ERROR (Status)) {  // Not definded yet so add it to the NV
Variables.
         // zero out buffer
        ZeroMem (&PrivateData->Configuration, sizeof
(MYWIZARDDRIVER_CONFIGURATION));
        Status = gRT->SetVariable(
                  mIfrVariableName,
                  &mMyWizardDriverFormSetGuid,
                  EFI_VARIABLE_NON_VOLATILE |
EFI_VARIABLE_BOOTSERVICE_ACCESS,
                  sizeof (MYWIZARDDRIVER_CONFIGURATION),
                  &PrivateData->Configuration   //  buffer is 000000
now
                  );
  }
```

17

```
270   }
271   Status = EFI_SUCCESS;
272
273   PrivateData->HiiHandle[0] = HiiHandle[0];
274
275   BufferSize = sizeof (MYWIZARDDRIVER_CONFIGURATION);
276
277   // IF driver is not part of the Platform then need to get/set defaults for t
278   Status = gRT->GetVariable (
279             mIfrVariableName,
280             &mMyWizardDriverFormSetGuid,
281             NULL,
282             &BufferSize,
283             &PrivateData->Configuration
284             );
285   if (EFI_ERROR (Status)) {  // Not definded yet so add it to the NV Variables
286             // zero out buffer
287         ZeroMem (&PrivateData->Configuration, sizeof (MYWIZARDDRIVER_CONFIGURA
288         Status = gRT->SetVariable(
289                   mIfrVariableName,
290                   &mMyWizardDriverFormSetGuid,
291                   EFI_VARIABLE_NON_VOLATILE | EFI_VARIABLE_BOOTSERVICE_ACCESS,
292                   sizeof (MYWIZARDDRIVER_CONFIGURATION),
293                   &PrivateData->Configuration   //  buffer is 000000  now
294                   );
295   }
296   //
297   // Install Driver Supported EFI Version Protocol onto ImageHandle
```

18

Save MyWizardDriver.c

19

Now onto the final file, MyWizardDriver.inf. **Add** the following protocols in the [protocols] section that are being used by copying and pasting (as shown below):

19

```
gEfiHiiStringProtocolGuid                    ## CONSUMES
gEfiHiiConfigRoutingProtocolGuid             ## CONSUMES
gEfiFormBrowser2ProtocolGuid                 ## CONSUMES
gEfiHiiDatabaseProtocolGuid                  ## CONSUMES
```

19

```
55   gEfiComponentNameProtocolGuid
56   gEfiHiiConfigAccessProtocolGuid
57   gEfiSimpleTextOutProtocolGuid
58
59
60   gEfiHiiStringProtocolGuid
61   gEfiHiiConfigRoutingProtocolGuid
62   gEfiFormBrowser2ProtocolGuid
63   gEfiHiiDatabaseProtocolGuid
64
```

20

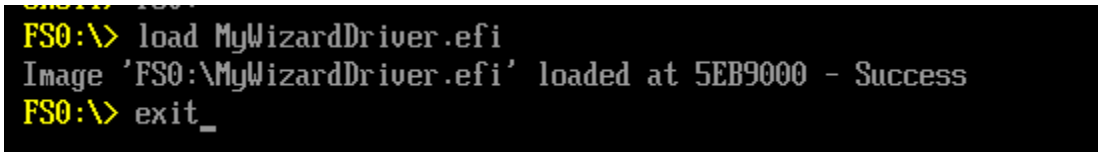
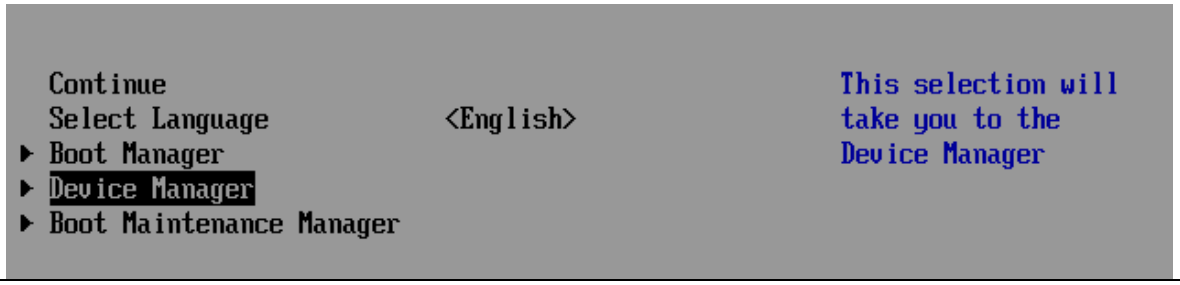
Save the MyWizardDriver.inf file. All the files should be saved at this point.

21

Re-Open the Visual Studio Command Prompt

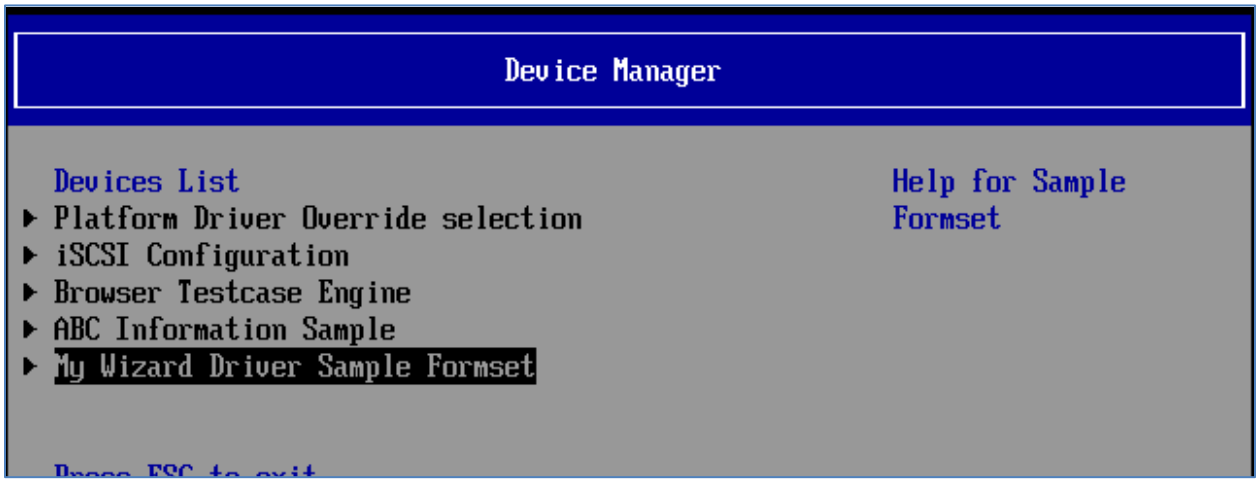
22

Add MyWizardDriver.inf to the Nt32Pkg.dsc (See Lab 8 building MyWizardDriver)

23	**Type** `build`
24	**Press** "Enter"
25	**Type** `build run`
26	**Press** "Enter"
27	**At the UEFI Shell prompt,type** `fs0:`
28	**Press** "Enter"
29	**Type** `Load MyWizardDriver.efi`
30	**Press** "Enter" This will load your driver into memory
31	 **Type** `exit`
32	**Press** "Enter"
33	Now at the setup front page menu, **select** "Device Manager" 
34	**Press** "Enter"

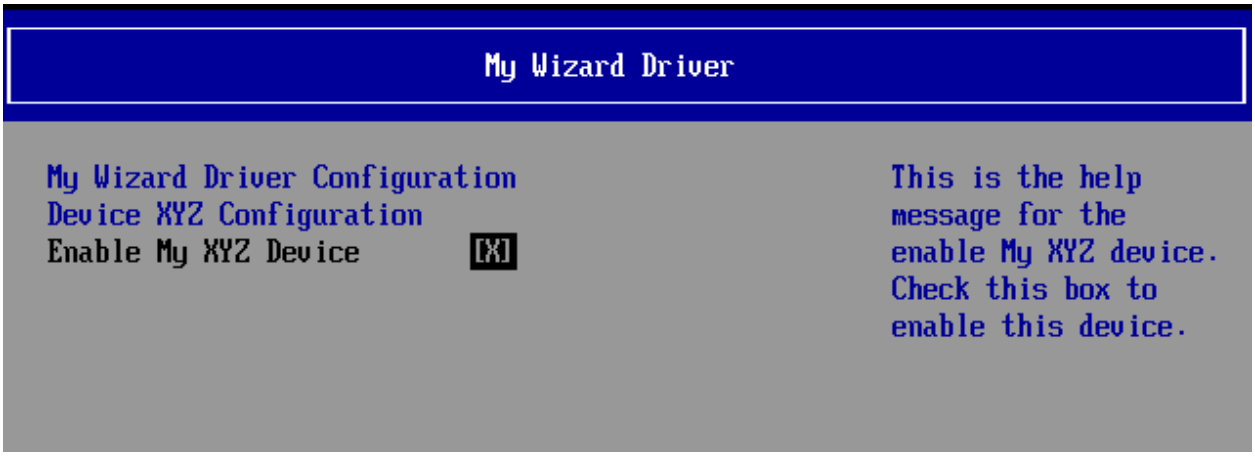
35

Inside the Device Manager menu **press** the down to “My Wizard Driver Sample Formset”



Press “Enter” .

36

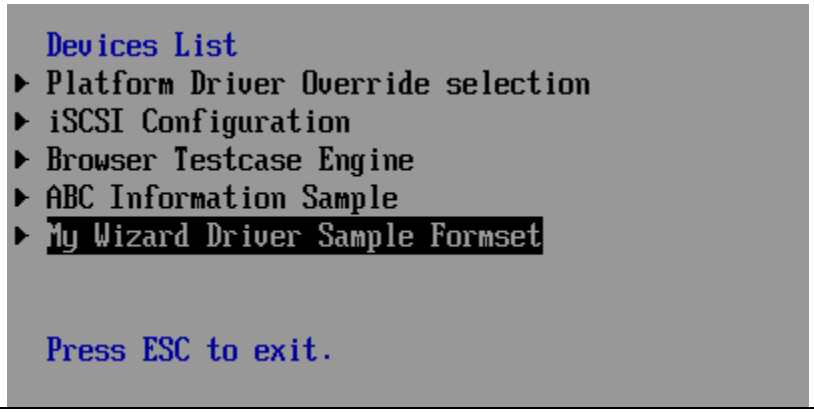
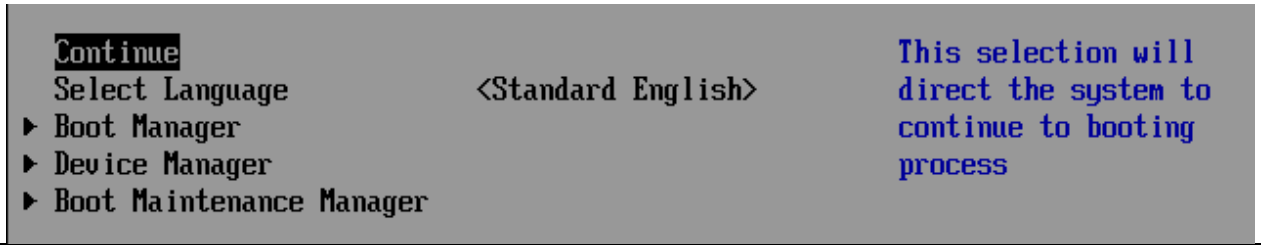


***Note*:** Notice that your form is now displayed with a choice to enable your device. Also notice the titles and help strings that are in the .UNI file you edited.

At this point since the HII configuration routing functions are not functional the values (Enable/ Disable) will not be saved to NVRAM. The next lab will update the HII Extract, Route, and call back functions for the HII configuration routing protocol your driver will produce.

37

Press the space bar to Enable and Disable the “Enable My XYZ Device”

38	**Press** F10 to attempt to save  ***Note***: You're not able to save the data changes at this point.
39	**Press** "Enter"
40	**Press** "Escape", and then "Y" to exit
41	To Exit the "Device Manager" Page: Press "Escape" 
42	**Press** Up Arrow to "Continue" 
43	**Press** "Enter"
44	At the Shell prompt **type** Reset 

45	**Press** "Enter" to return to the Visual Studio Command Prompt 

You've completed the first lab and added strings and forms to setup HII for user configuration. However, **the data is not saved to NVRAM**. In the next lab, you'll learn how to update HII to save data to NVRAM.

For any build issues copy the solution files from C:\Fw\LabSolutions\Lesson9.1

9.2 Updating HII to Save Data Settings

In this lab, you'll learn how to modify and update your driver's HII code to save the users settings into NVRAM. The UEFI Driver Wizard created the protocols for your driver to update and interface with the HII browser engine and database. The HII configuration access Protocol functions for MyWizardDriver are in the file C:\fw\edk2\MyWizardDriver\HiiConfigAccess.c. This next lab will install these protocols and update them to save the user data from the HII menus into NVRAM.

Step	Action
1	**Update** the MyWizardDriver.c file Your driver will need to keep track of the consumed protocols in it's own data structure so it will need to declare local pointers to these and then store them in its own private context data structure.
2	**Add** the following local variable declarations in the function `MyWizardDriverDriverEntryPoint` Entry Point (as shown below Approx. line 185):

```
EFI_HII_STRING_PROTOCOL          *HiiString;
EFI_FORM_BROWSER2_PROTOCOL       *FormBrowser2;
EFI_HII_CONFIG_ROUTING_PROTOCOL *HiiConfigRouting;
```

```
179   EFI_STATUS  Status;
180
181   // HII Locals
182   EFI_HII_PACKAGE_LIST_HEADER     *PackageListHeader;
183   EFI_HII_DATABASE_PROTOCOL       *HiiDatabase;
184   EFI_HII_HANDLE                   HiiHandle[2];
185   EFI_HII_STRING_PROTOCOL         *HiiString;
186   EFI_FORM_BROWSER2_PROTOCOL      *FormBrowser2;
187   EFI_HII_CONFIG_ROUTING_PROTOCOL *HiiConfigRouting;
188   EFI_STRING                       ConfigRequestHdr;
189   UINTN                            BufferSize;
190
191   Status = EFI_SUCCESS;
192
```

3	**Add** the following code to locate and store consumed protocols **before** the `// Publish sample Fromset and config access` comment (as shown below Approx. line 227): The reason is to Locate the Hii Database, Hii String, Hii Form browser and config routing protocols and store their pointers into the Private context data structure for your driver to access.

Step	Action

```
  //
  // Locate Hii Database protocol
  //
  Status = gBS->LocateProtocol (&gEfiHiiDatabaseProtocolGuid, NULL, (VOID **) &HiiDatabase);
  if (EFI_ERROR (Status)) {
    return Status;
  }
  PrivateData->HiiDatabase = HiiDatabase;

  //
  // Locate HiiString protocol
  //
  Status = gBS->LocateProtocol (&gEfiHiiStringProtocolGuid, NULL, (VOID **) &HiiString);
  if (EFI_ERROR (Status)) {
    return Status;
  }
  PrivateData->HiiString = HiiString;

  //
  // Locate Formbrowser2 protocol
  //
  Status = gBS->LocateProtocol (&gEfiFormBrowser2ProtocolGuid, NULL, (VOID **) &FormBrowser2);
  if (EFI_ERROR (Status)) {
    return Status;
  }
  PrivateData->FormBrowser2 = FormBrowser2;

  //
  // Locate ConfigRouting protocol
  //
  Status = gBS->LocateProtocol (&gEfiHiiConfigRoutingProtocolGuid, NULL, (VOID **)
&HiiConfigRouting);
  if (EFI_ERROR (Status)) {
    return Status;
  }
  PrivateData->HiiConfigRouting = HiiConfigRouting;
```

Step	Action

```
225
226
227   //
228   // Locate Hii Database protocol
229   //
230   Status = gBS->LocateProtocol (&gEfiHiiDatabaseProtocolGuid, NULL
231   if (EFI_ERROR (Status)) {
232     return Status;
233   }
234   PrivateData->HiiDatabase = HiiDatabase;
235
236   //
237   // Locate HiiString protocol
238   //
239   Status = gBS->LocateProtocol (&gEfiHiiStringProtocolGuid, NULL,
240   if (EFI_ERROR (Status)) {
241     return Status;
242   }
243   PrivateData->HiiString = HiiString;
244
245   //
246   // Locate Formbrowser2 protocol
247   //
248   Status = gBS->LocateProtocol (&gEfiFormBrowser2ProtocolGuid, NU
249   if (EFI_ERROR (Status)) {
250     return Status;
251   }
252   PrivateData->FormBrowser2 = FormBrowser2;
253
254   //
255   // Locate ConfigRouting protocol
256   //
257   Status = gBS->LocateProtocol (&gEfiHiiConfigRoutingProtocolGuid
258   if (EFI_ERROR (Status)) {
259     return Status;
260   }
261   PrivateData->HiiConfigRouting = HiiConfigRouting;
262
263
264   //
265   // Publish sample Fromset and config access
266   //
267   Status = gBS->InstallMultipleProtocolInterfaces (
```

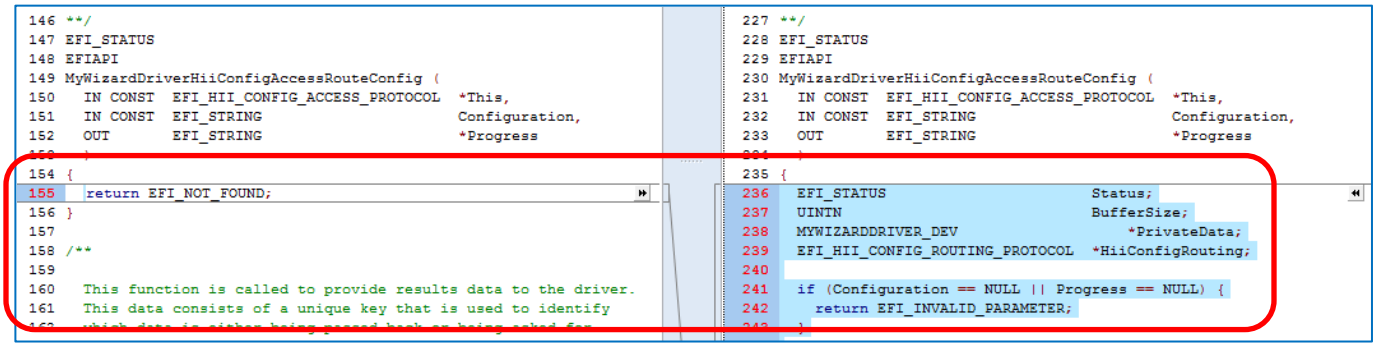
Step	Action
4	Since the Hii Database Protocol was located earlier in the code with the previous code insertion and is no longer necessary, **comment out** the old `OpenProtocol` code with the "//" (approx. lines 289-298, as shown below) and **add the comment** // Done above Make sure not to comment out the second " `if (!EFI_ERROR (Status)) {`" `Status = gBS->OpenProtocol (` `ImageHandle,` `&gEfiHiiPackageListProtocolGuid,` `(VOID **)&PackageListHeader,` `ImageHandle,` `NULL,` `EFI_OPEN_PROTOCOL_GET_PROTOCOL` `);` `// Done above` `// if (!EFI_ERROR (Status)) {` `//  //` `//  // Retrieve the pointer to the UEFI HII Database Protocol` `//  //` `//  Status = gBS->LocateProtocol (` `//  &gEfiHiiDatabaseProtocolGuid,` `//  NULL,` `//  (VOID **)&HiiDatabase` `//  );` `if (!EFI_ERROR (Status)) {` ***Note***: The earlier `LocateProtocol` code already found the pointer to the Hii Database protocol and stored it to the local pointer variable `HiiDatabase`. When we added the driver-consumed protocols, we searched via `LocateProtocol` for the Hii Database pointer function. Since we did it above we're now commenting out this code.
5	**Comment out** the **matching "}" with "//"** to the if statement (as shown below at approx. line 310): `if (!EFI_ERROR (Status)) {` `//` `// Register list of HII packages in the HII Database` `//` `Status = HiiDatabase->NewPackageList (` `HiiDatabase,` `PackageListHeader,` `mDriverHandle[0],` `&HiiHandle[0]` `);` `ASSERT_EFI_ERROR (Status);` `// }` `}` `Status = EFI_SUCCESS;`
6	**Save** MyWizardDriver.c

Step	Action
7	**Open** C:\fw\edk2\MyWizardDriver\HiiConfigAccess.c**.** The Driver Wizard only made dummy functions for the extract, route and callback functions. In order to save the Data passed into the forms from the Hii Browser engine, you will need to port these functions to be functional.
8	**Add** the following `extern` statements for the form GUID and the NVRam variable (as shown below) these are global to the driver module only hence the beginning lower case "`m`" is the standard for a global for a module : `extern EFI_GUID mMyWizardDriverFormSetGuid;` `extern CHAR16 mIfrVariableName[];` 12 #include "MyWizardDriver.h" 13 14 extern EFI_GUID mMyWizardDriverFormSetGuid; 15 extern CHAR16 mIfrVariableName[]; 16 17 18 /// 19 /// HII Config Access Protocol instance
9	**Locate** `MyWizardDriverHiiConfigAccessExtractConfig` and **replace** line 108, "**`return EFI_NOT_FOUND`**", with the following code spread over **two** pages: FROM: 95 EFI_STATUS 96 EFIAPI 97 MyWizardDriverHiiConfigAccessExtractConfig (98 IN CONST EFI_HII_CONFIG_ACCESS_PROTOCOL *This, 99 IN CONST EFI_STRING Request, 100 OUT EFI_STRING *Progress, 101 OUT EFI_STRING *Results 102) 103 { 104 return EFI_NOT_FOUND; 105 } 106 107 /** TO: 99 EFI_STATUS 100 EFIAPI 101 MyWizardDriverHiiConfigAccessExtractConfig (102 IN CONST EFI_HII_CONFIG_ACCESS_PROTOCOL *This, 103 IN CONST EFI_STRING Request, 104 OUT EFI_STRING *Progress, 105 OUT EFI_STRING *Results 106) 107 { 108 EFI_STATUS Status; 109 UINTN BufferSize; 110 MYWIZARDDRIVER_DEV *PrivateData; 111 EFI_HII_CONFIG_ROUTING_PROTOCOL *HiiConfigRouting

Step	Action

```
  EFI_STATUS                      Status;
  UINTN                           BufferSize;
  MYWIZARDDRIVER_DEV                  *PrivateData;
  EFI_HII_CONFIG_ROUTING_PROTOCOL  *HiiConfigRouting;
  EFI_STRING                      ConfigRequest;
  EFI_STRING                      ConfigRequestHdr;
  UINTN                           Size;
  BOOLEAN                         AllocatedRequest;
  if (Progress == NULL || Results == NULL) {
    return EFI_INVALID_PARAMETER;
  }
  //
  // Initialize the local variables.
  //
  ConfigRequestHdr  = NULL;
  ConfigRequest     = NULL;
  Size              = 0;
  *Progress         = Request;
  AllocatedRequest  = FALSE;
  PrivateData = MYWIZARDDRIVER_DEV_FROM_THIS (This);
  HiiConfigRouting = PrivateData->HiiConfigRouting;
  //
  // Get Buffer Storage data from EFI variable.
  // Try to get the current setting from variable.
  //
  BufferSize = sizeof (MYWIZARDDRIVER_CONFIGURATION);
  Status = gRT->GetVariable (
            mIfrVariableName,
            &mMyWizardDriverFormSetGuid,
            NULL,
            &BufferSize,
            &PrivateData->Configuration
       );
```

Step	Action
	`if (EFI_ERROR (Status)) {` `  return EFI_NOT_FOUND;` `}` `  if (Request == NULL) {` `  DEBUG ((DEBUG_INFO, "\n:: Inside of Extract Config and Request == Null "));` `  } else {` `  ConfigRequest = Request;` `  }` `  //` `  // Convert buffer data to <ConfigResp> by helper function BlockToConfig()` `  //` `  Status = HiiConfigRouting->BlockToConfig (` `  HiiConfigRouting,` `  ConfigRequest,` `  (UINT8 *) &PrivateData->Configuration,` `  BufferSize,` `  Results,` `  Progress` `  );` `  //` `  // Free the allocated config request string.` `  //` `  if (AllocatedRequest) {` `  FreePool (ConfigRequest);` `  }` `  //` `  // Set Progress string to the original request string.` `  //` `  if (Request == NULL) {` `  *Progress = NULL;` `  } else if (StrStr (Request, L"OFFSET") == NULL) {` `  *Progress = Request + StrLen (Request);` `  }` `  return Status;`
10	Now **locate** `MyWizardDriverHiiConfigAccessRouteConfig` and **replace** line at approx. 228, **“return EFI_NOT_FOUND”**, with the following code:

Step	Action
	FROM: TO: 

```
146 **/
147 EFI_STATUS
148 EFIAPI
149 MyWizardDriverHiiConfigAccessRouteConfig (
150   IN CONST  EFI_HII_CONFIG_ACCESS_PROTOCOL  *This,
151   IN CONST  EFI_STRING                      Configuration,
152   OUT       EFI_STRING                      *Progress
153   )
154 {
155   return EFI_NOT_FOUND;
156 }
157
158 /**
159
160   This function is called to provide results data to the driver.
161   This data consists of a unique key that is used to identify
```

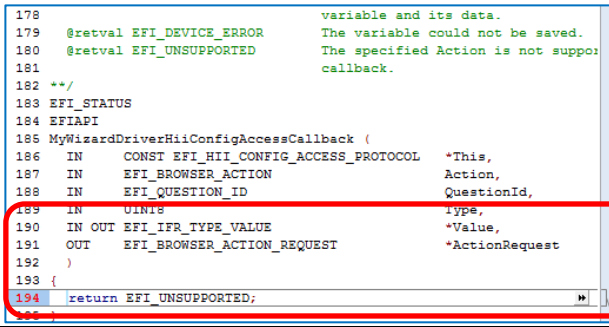
```
227 **/
228 EFI_STATUS
229 EFIAPI
230 MyWizardDriverHiiConfigAccessRouteConfig (
231   IN CONST  EFI_HII_CONFIG_ACCESS_PROTOCOL  *This,
232   IN CONST  EFI_STRING                      Configuration,
233   OUT       EFI_STRING                      *Progress
234   )
235 {
236   EFI_STATUS                    Status;
237   UINTN                         BufferSize;
238   MYWIZARDDRIVER_DEV            *PrivateData;
239   EFI_HII_CONFIG_ROUTING_PROTOCOL  *HiiConfigRouting;
240
241   if (Configuration == NULL || Progress == NULL) {
242     return EFI_INVALID_PARAMETER;
243   }
```

Step	Action

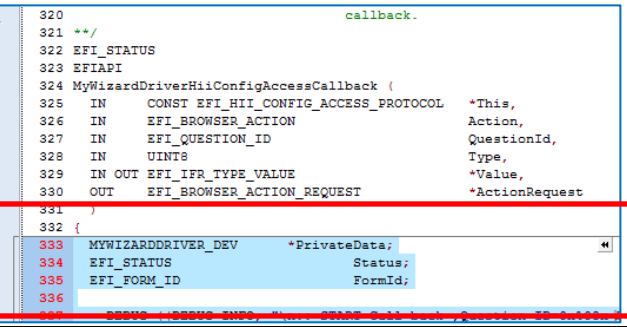
```
  EFI_STATUS                       Status;
  UINTN                            BufferSize;
  MYWIZARDDRIVER_DEV               *PrivateData;
  EFI_HII_CONFIG_ROUTING_PROTOCOL  *HiiConfigRouting;

  if (Configuration == NULL || Progress == NULL) {
    return EFI_INVALID_PARAMETER;
  }

  PrivateData = MYWIZARDDRIVER_DEV_FROM_THIS (This);
  HiiConfigRouting = PrivateData->HiiConfigRouting;
  *Progress = Configuration;

  //
  // Get Buffer Storage data from EFI variable
  //
  BufferSize = sizeof (MYWIZARDDRIVER_CONFIGURATION);
  Status = gRT->GetVariable (
            mIfrVariableName,
            &mMyWizardDriverFormSetGuid,
            NULL,
            &BufferSize,
            &PrivateData->Configuration
            );
  if (EFI_ERROR (Status)) {
    return Status;
  }

  //
  // Convert <ConfigResp> to buffer data by helper function ConfigToBlock()
  //
  BufferSize = sizeof (MYWIZARDDRIVER_CONFIGURATION);
  Status = HiiConfigRouting->ConfigToBlock (
                               HiiConfigRouting,
                               Configuration,
                               (UINT8 *) &PrivateData->Configuration,
                               &BufferSize,
                               Progress
                               );
  if (EFI_ERROR (Status)) {
    return Status;
  }

  //
  // Store Buffer Storage back to EFI variable
  //
  Status = gRT->SetVariable(
                  mIfrVariableName,
                  &mMyWizardDriverFormSetGuid,
                  EFI_VARIABLE_NON_VOLATILE | EFI_VARIABLE_BOOTSERVICE_ACCESS,
                  sizeof (MYWIZARDDRIVER_CONFIGURATION),
                  &PrivateData->Configuration
                  );
          DEBUG ((DEBUG_INFO, "\n:: ROUTE CONFIG Saving the configuration to NVRAM \n"));

  return Status;


  //return EFI_NOT_FOUND;
```

Step	Action
11	Lastly, **locate** MyWizardDriverHiiConfigAccessCallback and **replace** at approx. line 326, "**return EFI_UNSUPPORTED;**", with the following code:

FROM:

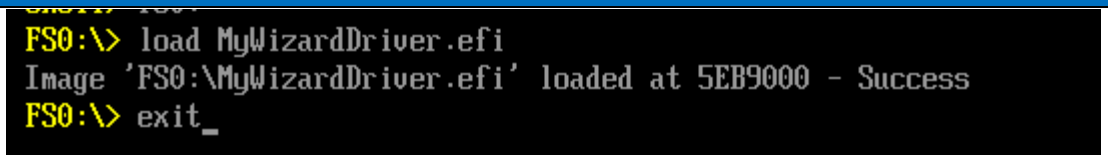
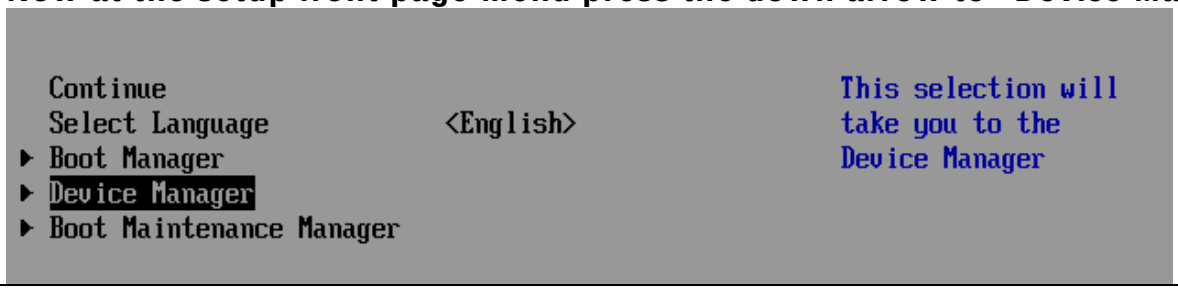
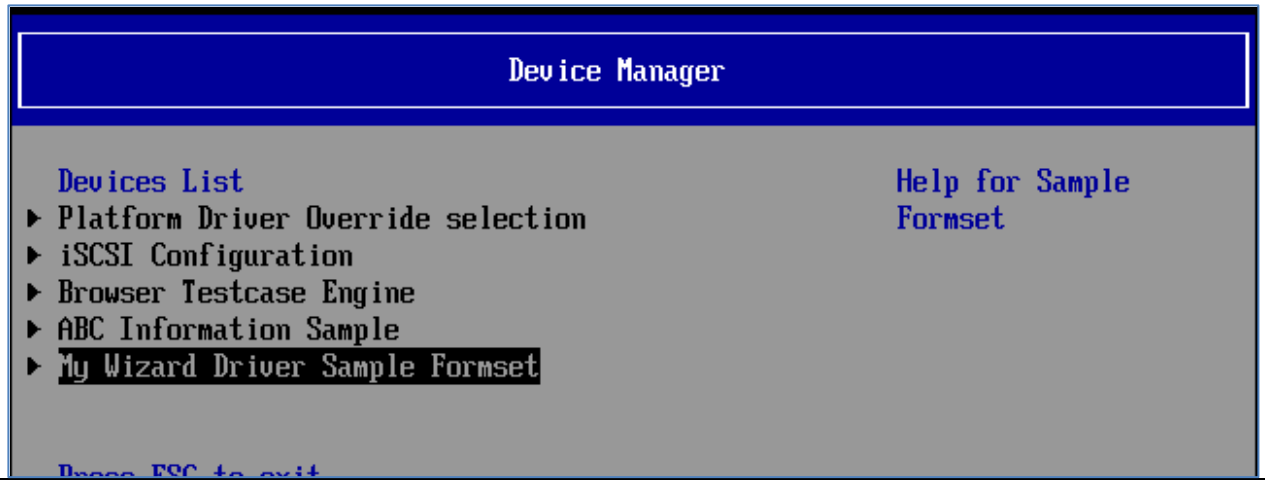
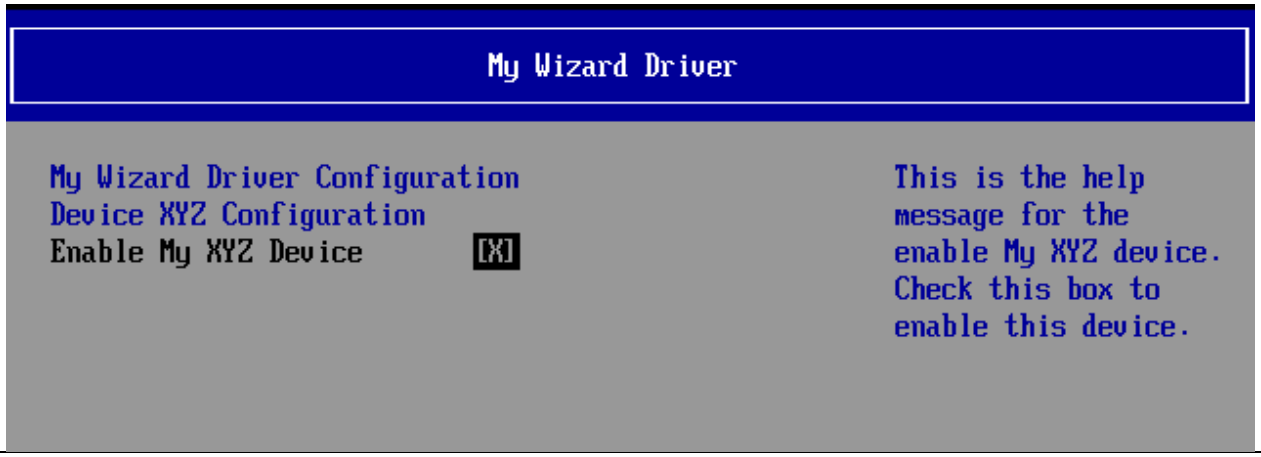


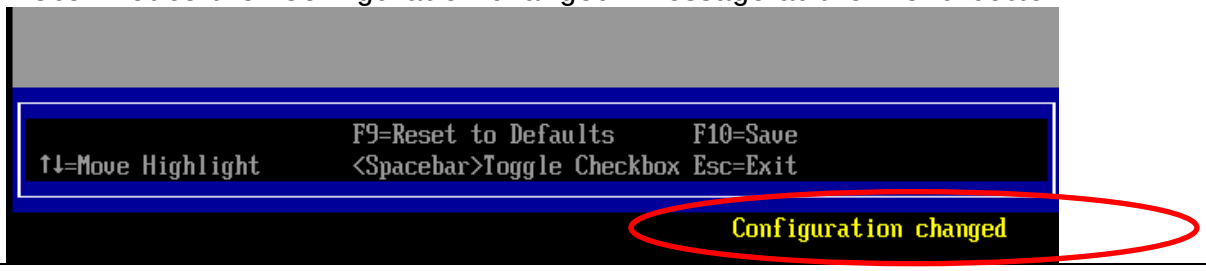
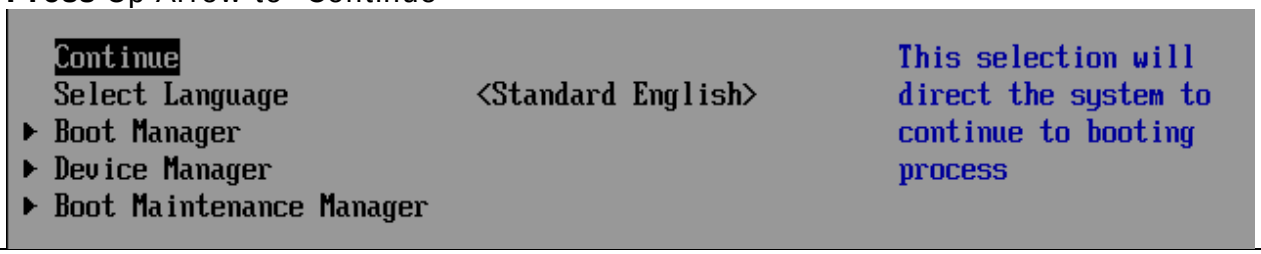
TO:

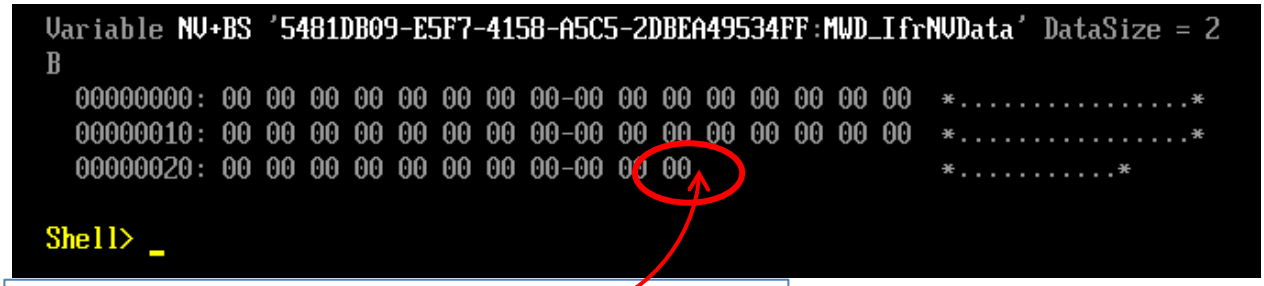
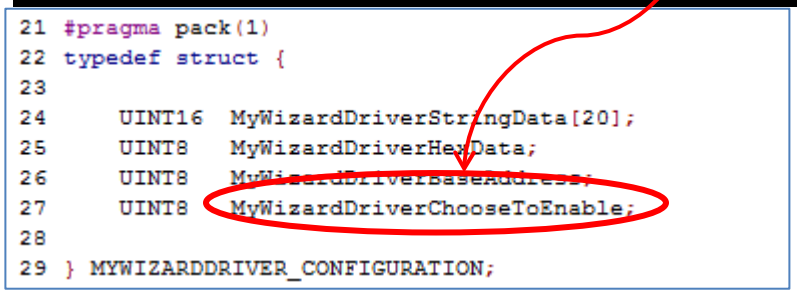
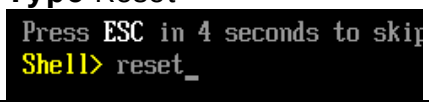



```
  MYWIZARDDRIVER_DEV      *PrivateData;
  EFI_STATUS                      Status;
  EFI_FORM_ID                     FormId;

    DEBUG ((DEBUG_INFO, "\n:: START Call back ,Question ID=0x%08x Type=0x%04x Action=0x%04x",
QuestionId, Type, Action));



  if (((Value == NULL) && (Action != EFI_BROWSER_ACTION_FORM_OPEN) && (Action !=
EFI_BROWSER_ACTION_FORM_CLOSE))||
    (ActionRequest == NULL)) {
    return EFI_INVALID_PARAMETER;
  }


  FormId = 0;
  Status = EFI_SUCCESS;
  PrivateData = MYWIZARDDRIVER_DEV_FROM_THIS (This);

  return Status;
```

12	**Save** HiiConfigAccess.c
13	In the Visual Studio Command Prompt, **type** `build`
14	**Press** "Enter"
15	**Type** `build run`
16	**Press** "Enter"
17	**At the UEFI Shell prompt, type** `fs0:`
18	**Press** "Enter"
19	**Type** `Load MyWizardDriver.efi`
20	**Press** "Enter"

Step	Action
21	 **Type** exit
22	**Press** "Enter"
23	**Now at the setup front page menu press the down arrow to "Device Manager"** 
24	**Press** "Enter"
25	Inside the Device Manager menu **select** "My Wizard Driver Sample Formset" 
26	**Press** "Enter" . 

Step	Action
27	***Note***: Once you hit "Enter", notice that your form is now displayed with a choice to enable your Device. Also notice the titles and help strings that are in the .UNI file you edited.
28	Test by **Press** the space bar to Enable and Disable the "Enable My XYZ Device" to change its value from [X] to [ ]
29	**Note**: Notice the "Configuration changed" message at the menu bottom. 
30	**Press** "F10"
31	**Press** "Escape" to exit
32	**Press** "Escape" to exit the "Device Manager" Page
33	**Press** Up Arrow to "Continue" 
34	**Press** "Enter"
35	At the Shell Prompt **type** `dmpstore -all`

Step	Action
36	**Notice** that enable is selected and saved in NVRam as the value of 0x00:   Because our data structure in MyWizardDriverNVDataStruc.h is stored in NVRAM with the variable name `MWD_IfrNVData` of type `MYWIZARDDRIVER_CONFIGURATION`, we can see the changes from our menu accessing through our HII forms. Notice that the enable/disable byte is the last byte in data structure `MWD_IfrNVData.MyWizardDriverChooseToEnable` where `00 ==` disable and `01 ==` enable.
37	**Type** Reset 
38	**Press** "Enter" to return to the Visual Studio Command Prompt 

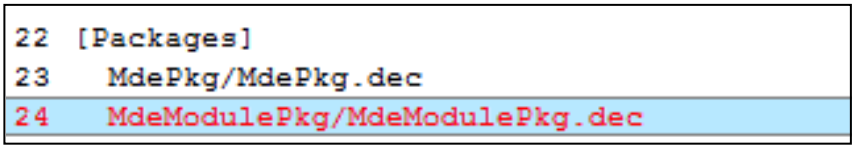
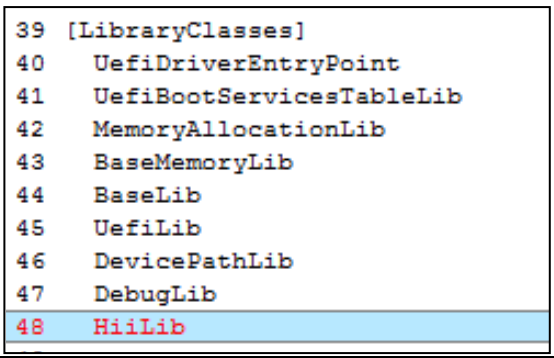
For any build issues copy the solution files from C:\Fw\LabSolutions\Lesson9.2
NOTE: Del Directory C:\fw\edk2\Build\NT32\DEBUG_VS2010x86\IA32\MyWizardDriver before the Build command to build the MyWizardDriver Clean

9.3 Updating your driver to initialize data from the VFR data to the HII Database

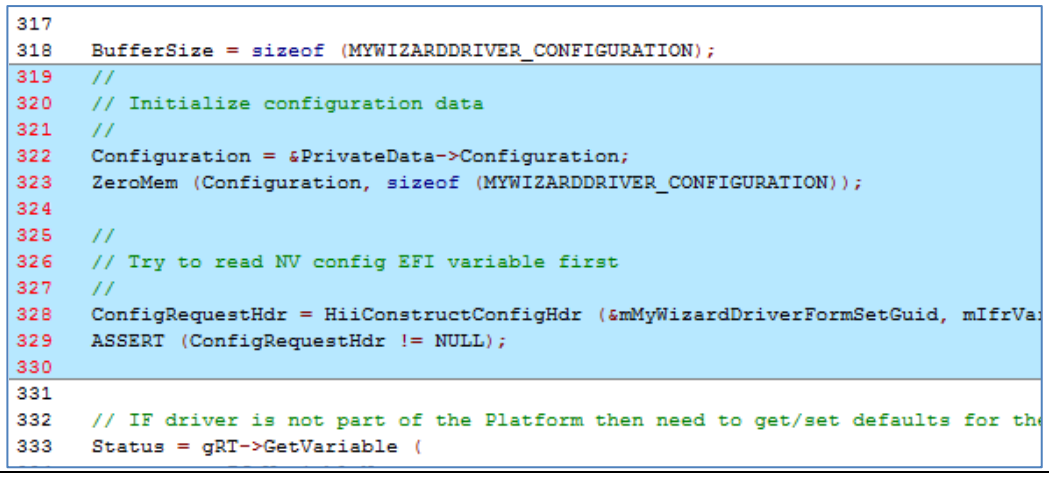
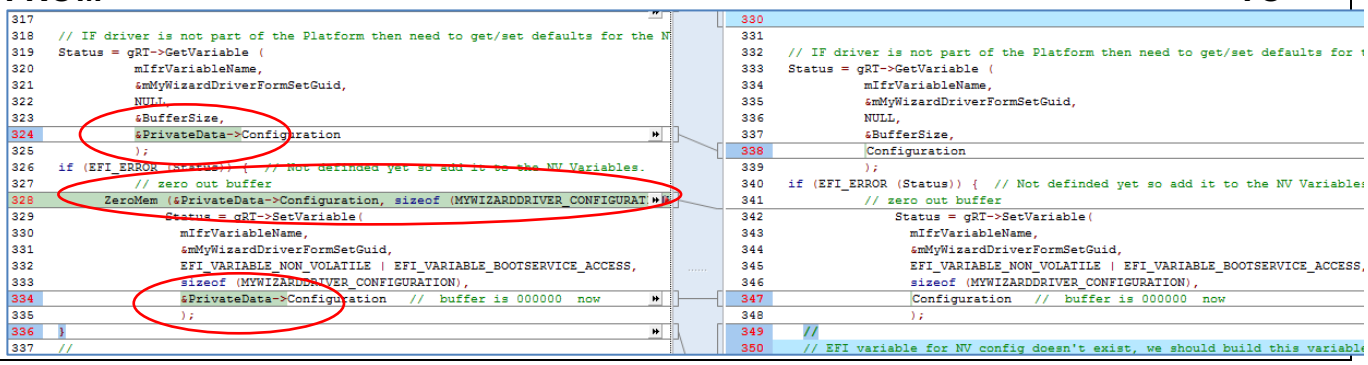
In this lab, you'll learn how to update your driver to initialize the data according to the defaults set in the .VFR file. Thus when the user enters your driver's menu for the first time, the values will display the defaults according to the .VFR file settings. You will also learn the rich set of HII function calls that are part of the MdeModulePkg in the HiiLib by reviewing the "MdeModulePkg Document.chm".

9.3.1 *Add HII Library Calls to Your Driver*

For this lab you will update the following files: MyWizardDriver.inf, MyWizardDriver.h, and MyWizardDriver.c

Step	Action
1	**Update** the MyWizardDriver.inf file
2	**Add** the following package (as shown below): The HII Library in the MdeModulePkg has many functions to help with Communication to/from the Hii Database and Hii forms. One function call `HiiSetToDefaults` will compare the default settings from the .VFR file and update the driver's configuration buffer according to the settings in the .VFR file.
	`MdeModulePkg/MdeModulePkg.dec`
	 ***Note***: For other functions from the HII Library, open the .chm file "MdeModulePkg Document.chm" and search for HiiLib.h.
3	**Add** the following library class (as shown below):
	`HiiLib`
	

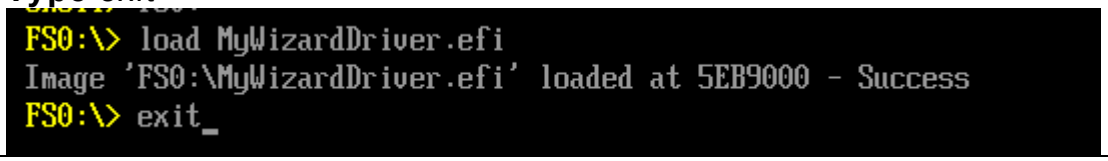
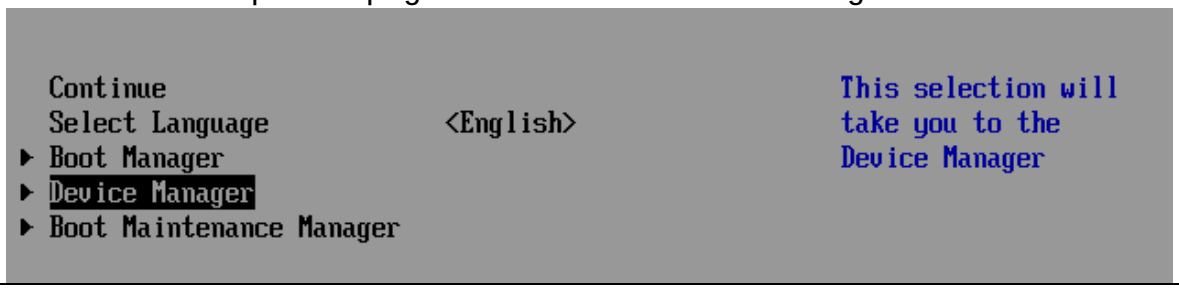
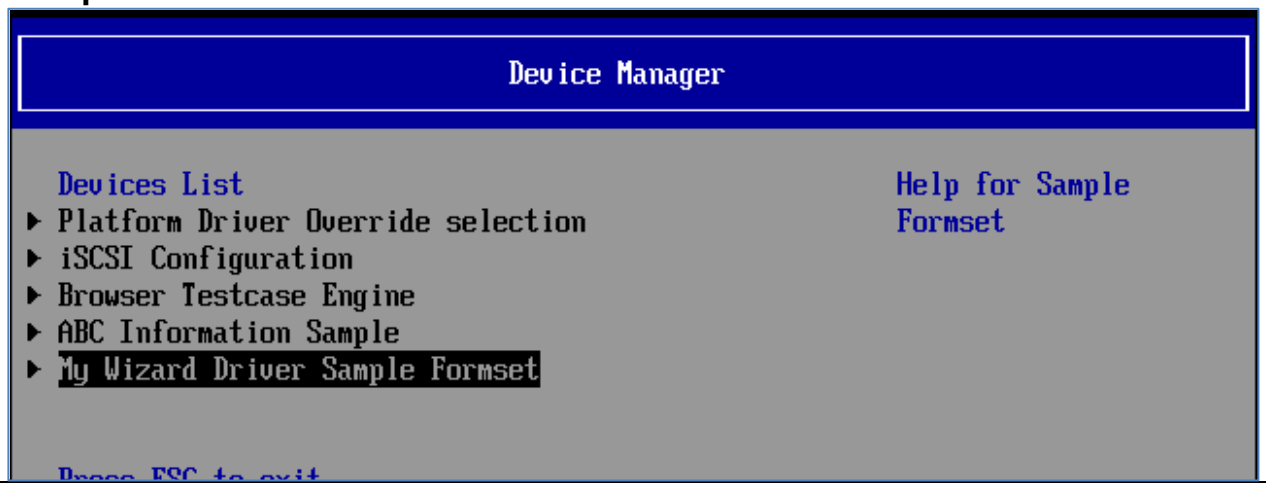
Step	Action
4	**Save** MyWizardDriver.inf
5	**Update** the MyWizardDriver.h file
6	**Add** the following code (as shown below): `#include <Library/HiiLib.h>` `42 // Added for HII` `43 #include <Protocol/HiiConfigRouting.h>` `44 #include <Protocol/FormBrowser2.h>` `45 #include <Protocol/HiiString.h>` `46 #include <Library/DevicePathLib.h>` `47 #include <Library/HiiLib.h>`
7	**Save** MyWizardDriver.h
8	**Update** the MyWizardDriver.c file
9	**Add Locals:** first add 2 locals for your drivers configuration buffer and a boolean flag from the Hii Library calls Add the following at Approx. Line 190. `MYWIZARDDRIVER_CONFIGURATION *Configuration;` `BOOLEAN ActionFlag;` `180` `181 // HII Locals` `182 EFI_HII_PACKAGE_LIST_HEADER *PackageListHeader;` `183 EFI_HII_DATABASE_PROTOCOL *HiiDatabase;` `184 EFI_HII_HANDLE HiiHandle[2];` `185 EFI_HII_STRING_PROTOCOL *HiiString;` `186 EFI_FORM_BROWSER2_PROTOCOL *FormBrowser2;` `187 EFI_HII_CONFIG_ROUTING_PROTOCOL *HiiConfigRouting;` `188 EFI_STRING ConfigRequestHdr;` `189 UINTN BufferSize;` `190 MYWIZARDDRIVER_CONFIGURATION *Configuration;` `191 BOOLEAN ActionFlag;` `192` `193 Status = EFI_SUCCESS;` `194`
10	**Add** the following to the MyWizardDriverDriverEntryPoint entry point funtion to line 319, approximately after “`BufferSize =`” as shown below

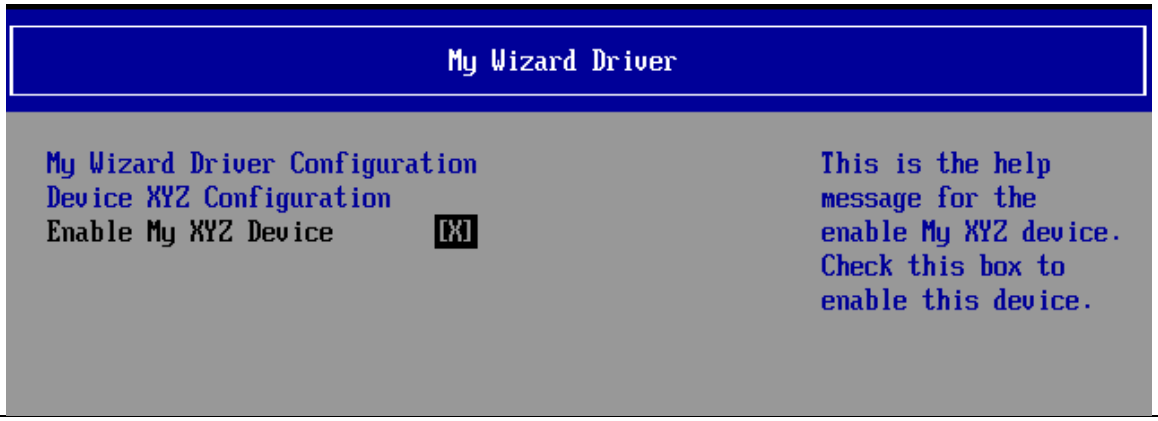
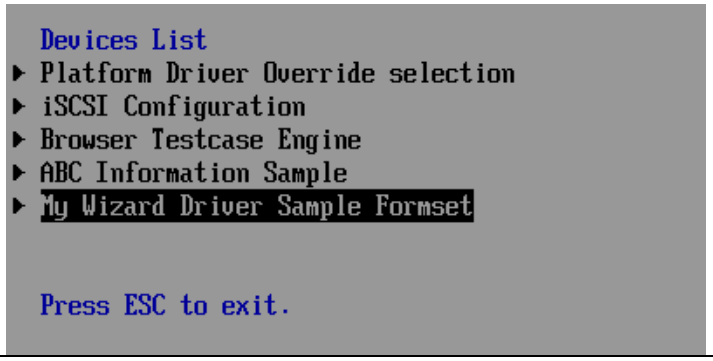
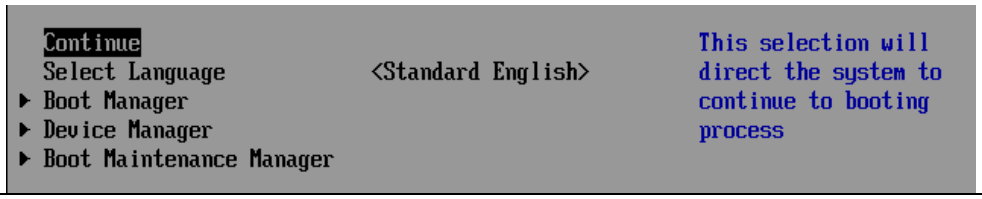
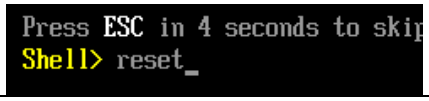
Step	Action
	`//` `// Initialize configuration data` `//` `Configuration = &PrivateData->Configuration;` `ZeroMem (Configuration, sizeof (MYWIZARDDRIVER_CONFIGURATION));` `//` `// Try to read NV config EFI variable first` `//` `ConfigRequestHdr = HiiConstructConfigHdr (&mMyWizardDriverFormSetGuid, mIfrVariableName, mDriverHandle[0]);` `ASSERT (ConfigRequestHdr != NULL);`
	
11	Modify the following lines: @~338: remove: “&PrivateData->” from the “&PrivateData->Configuration” @~342: remove line: ZeroMem (&PrivateData->Configuration, sizeof (MYWIZARDDRIVER_CONFIGURATION)); @~347: remove: “&PrivateData->” from the “&PrivateData->Configuration”
	FROM **TO** 

Step	Action
12	**Add** the following code to the `MyWizardDriverDriverEntryPoint` entry point code at approximately line 349 before `//` `// Install Driver Supported EFI Version Protocol onto ImageHandle` `//` You're **deleting** the "}' and replacing it with the following code (as shown below). With this replacement we are adding an "`else`" to the "`if`" statement:

```
    //
    // EFI variable for NV config doesn't exist, we should build this
variable
    // based on default values stored in IFR
    //
    ActionFlag = HiiSetToDefaults (ConfigRequestHdr,
EFI_HII_DEFAULT_CLASS_STANDARD);
    ASSERT (ActionFlag);
  } else {
    //
    // EFI variable does exist and Validate Current Setting
    //
    ActionFlag = HiiValidateSettings (ConfigRequestHdr);
    ASSERT (ActionFlag);
  }  // Match if (EFI_ERROR (Status))
  FreePool (ConfigRequestHdr);

// end HII
```

```
347                     Configuration   //  buffer is 000000  now
348                     );
349     //
350     // EFI variable for NV config doesn't exist, we should build this variable
351     // based on default values stored in IFR
352     //
353     ActionFlag = HiiSetToDefaults (ConfigRequestHdr, EFI_HII_DEFAULT_CLASS_STAN
354     ASSERT (ActionFlag);
355   } else {
356     //
357     // EFI variable does exist and Validate Current Setting
358     //
359     ActionFlag = HiiValidateSettings (ConfigRequestHdr);
360     ASSERT (ActionFlag);
361   }  // Match if (EFI_ERROR (Status))
362   FreePool (ConfigRequestHdr);
363
364
365   // end HII
366   //
367   // Install Driver Supported EFI Version Protocol onto ImageHandle
368   //
369   Status = gBS->InstallMultipleProtocolInterfaces (
```

Step	Action
	Note the "}" on line 361 is still matching the initial if statement. Make sure you do not have a duplicate "}"
13	**Save** the MyWizardDriver.c file
14	In the Visual Studio Command Prompt, **type** build
15	**Press** "Enter"
16	**Type** build run
17	**Press** "Enter"
18	At the UEFI Shell prompt, **type** fs0:
19	**Press** "Enter"
20	**Type** Load MyWizardDriver.efi
21	**Press** "Enter"
22	**Type** exit 
23	**Press** "Enter"
24	Now at the setup front page menu **select** "Device Manager" 
25	**Press** "Enter"
	Inside the Device Manager menu press the down arrow to **"My Wizard Driver Sample Formset"** 

Step	Action
26	**Press** “Enter” . 
27	**Press** “Escape” to exit
28	To Exit the “Device Manager” Page: **Press** “Escape” 
29	**Press** Up Arrow to **“Continue”** 
30	**Press** “Enter”
31	**Type** Reset 

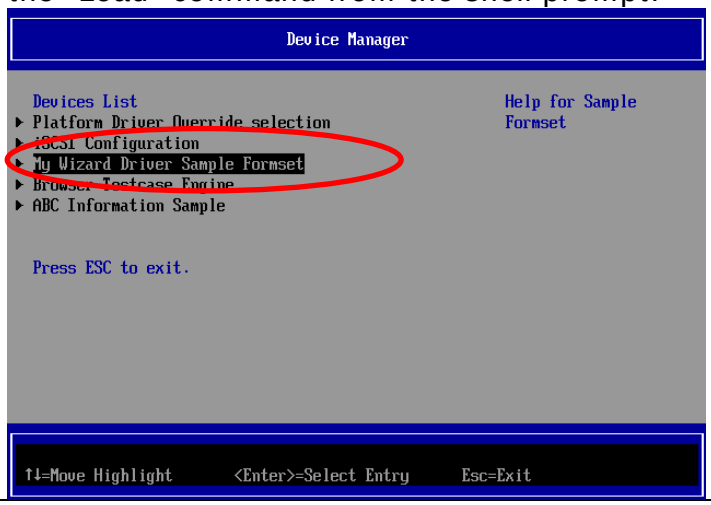
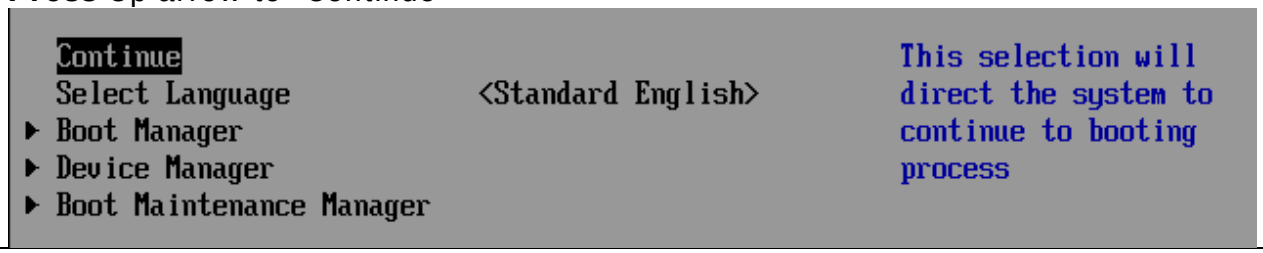
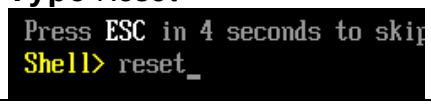
Step	Action
32	**Press** "Enter" to return to the Visual Studio Command Prompt 

For any build issues copy the solution files from C:\Fw\LabSolutions\Lesson9.3
NOTE: Del Directory C:\fw\edk2\Build\NT32\DEBUG_VS2010x86\IA32\MyWizardDriver before the Build command to build the MyWizardDriver Clean

9.3.2 *Add your Driver to the platform*

As of now, your driver needs to be soft loaded each time from the shell prompt. In this lab, you'll update the platform .FDF file to force your driver to load as part of the platform UEFI driver.

Step	
1	**Open** to update: C:\fw\edk2\Nt32PkgNt32Pkg.Fdf A**dd** the following code (as shown below before "`!if $(BUILD_NEW_SHELL) == TRUE`"):
2	INF MyWizardDriver/MyWizardDriver.inf `INF MdeModulePkg/Universal/Network/IScsiDxe/IScsiDxe.inf` `INF MyWizardDriver/MyWizardDriver.inf` `!if $(BUILD_NEW_SHELL) == TRUE` `INF ShellPkg/Application/Shell/Shell.inf` `!endif`
3	**Save** Nt32pkg.fdf
4	In the Visual Studio Command Prompt, **type** `build`
5	**Press** "Enter"
6	**Type** `build run`
7	**Press** "Enter"
8	**At the Shell prompt type:** `exit` `Press ESC in 1 seconds to skip` `Shell> exit_`
9	**Press** "Enter"
10	Now at the setup front page menu press the down arrow to **"Device Manager"** Continue Select Language <English> ▶ Boot Manager ▶ Device Manager ▶ Boot Maintenance Manager This selection will take you to the Device Manager
11	**Press** "Enter"

11	**Notice** that the `My Wizard Driver Sample Formset` is added without having to issue the "Load" command from the shell prompt. 
12	**Press** "Escape"
13	**Press** Up arrow to "Continue" 
14	**Press "Enter"**
15	**Type** Reset 
16	**Press** "Enter" to return to the Visual Studio Command Prompt 

9.4 Updating the Menu: Reset Button

In this lab, you'll learn how to add a reset button to your driver's form menu. It's time to add more configuration fields to your menu, enabling users to modify more fields now that you've built a driver that 1) saves data from forms into NVRAM 2) updates data from the .VFR forms and 3) builds into the platform drivers.

The next set of labs will update .VFR, MyWizardDriver.vfr, and UNI MyWizardDriver.uni string files to incrementally add a reset button (9.4), pop-up box (9.5), string name (9.6), and numeric hex value (9.7) to your driver's form menu:

Step	Action
1	**Update** the MyWizardDriver.vfr file
2	**Add** the following code (as shown below after the "GUID" definition Apprx. Line 29): With this code you are created a VFR sub-function called "MyStandardDefault"

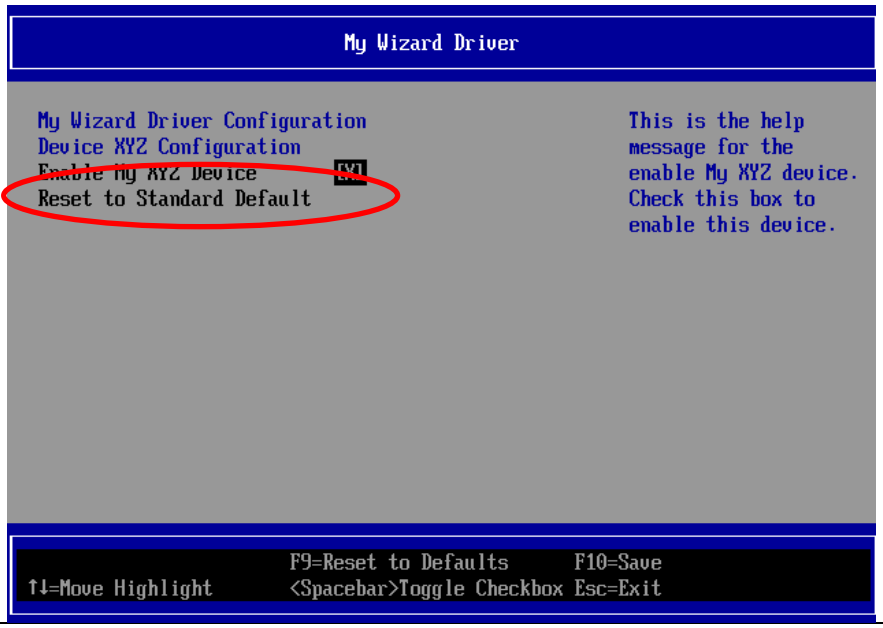
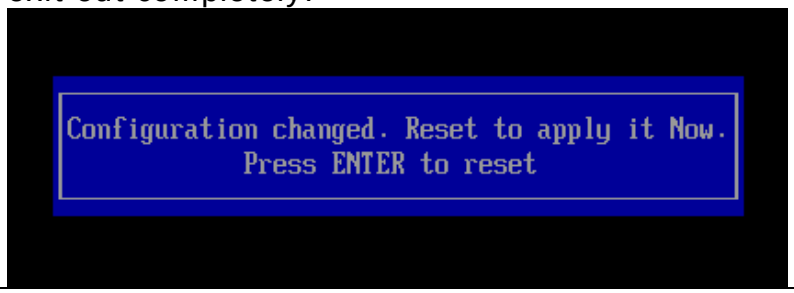
```
defaultstore MyStandardDefault,
    prompt      = STRING_TOKEN(STR_STANDARD_DEFAULT_PROMPT),
    attribute   = 0x0000;        // Default ID: 0000 standard
default
```

```
27     guid  = MYWIZARDDRIVER_FORMSET_GUID;    // GUID of this buffer storage
28
29 defaultstore MyStandardDefault,
30     prompt      = STRING_TOKEN(STR_STANDARD_DEFAULT_PROMPT),
31     attribute   = 0x0000;                        // Default ID: 0000 standard default
32
```

3	**Add** the folowing code before the "endform" (as shown below Approx. Line 55):

```
    resetbutton
      defaultstore = MyStandardDefault,
      prompt    =
STRING_TOKEN(STR_STANDARD_DEFAULT_PROMPT_RESET),
      help      = STRING_TOKEN(STR_STANDARD_DEFAULT_HELP),
    endresetbutton;
```

Step	Action
| | ```
52      endcheckbox;
53
54
55    resetbutton
56      defaultstore = MyStandardDefault,
57      prompt    = STRING_TOKEN(STR_STANDARD_DEFAULT_PROMPT_RESET),
58      help      = STRING_TOKEN(STR_STANDARD_DEFAULT_HELP),
59    endresetbutton;
60
61
62  endform;
63
64 endformset;
```<br>**Save** MyWizardDriver.vfr |
4	**Save** MyWizardDriver.vfr
5	**Update** the MyWizardDriver.uni file
6	**Add** the following strings at the end of the file to support the "**STR**_" referenced added in the .vfr file: `#string STR_STANDARD_DEFAULT_PROMPT    #language en "Standard Default"` `#string STR_STANDARD_DEFAULT_PROMPT_RESET    #language en "Reset to Standard Default"` `#string STR_STANDARD_DEFAULT_HELP      #language en "This will reset all the Questions to their standard default value"`
7	**Save** MyWizardDriver.uni
8	In the Visual Studio Command Prompt, **type** `build`
9	**Press** "Enter"
10	**Type** `build run`
11	**Press** "Enter"
12	**Type** exit
13	**Press** "Enter"
14	Now at the setup front page menu press the **down arrow** to **"Device Manager"**
15	**Press** "Enter"
16	Inside the Device Manager menu press the down arrow to **"My Wizard Driver Sample Formset"**
```

Step	Action
17	**Access** the My Wizard Driver menu and **notice** the item "Reset to Standard Default" 
18	**Press Down Arrow to "**Reset to Standard Default"
19	**Press** "Enter
20	**Notice** the **"Configuration changed"** message at the bottom of the menu
21	**To Exit Press** "Escape" then "Y"
22	**To Exit the "Device Manager" Page: Press** "Escape"
23	**Press** Up Arrow to **"Continue"**
24	**Observe:** Notice that since this change requires a reset, the Nt32 will exit out completely. 
25	**Press** "Enter" to return to the Visual Studio Command Prompt 

For any build issues copy the solution files from C:\Fw\LabSolutions\Lesson9.4
NOTE: Del Directory C:\fw\edk2\Build\NT32\DEBUG_VS2010x86\IA32\MyWizardDriver before the Build command to build the MyWizardDriver Clean.

9.5 Updating the Menu: Pop-up Box

In this lab, you'll learn how to add a *pop-up box* to your driver's form menu by using the "**oneof**" VFR term. We will also only update the MyWizardDriver.vfr and MyWizardDriver.uni files.

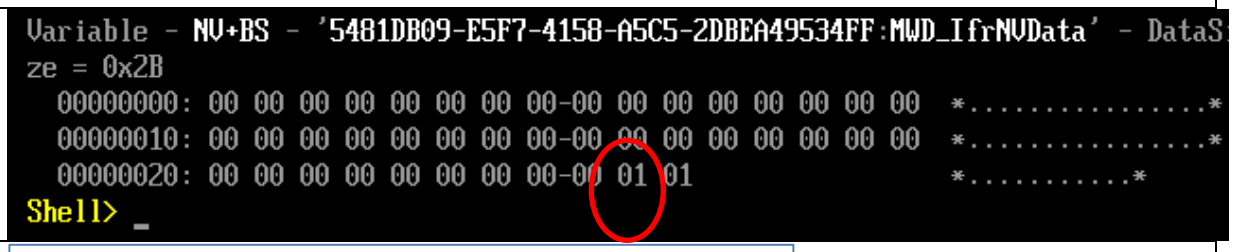
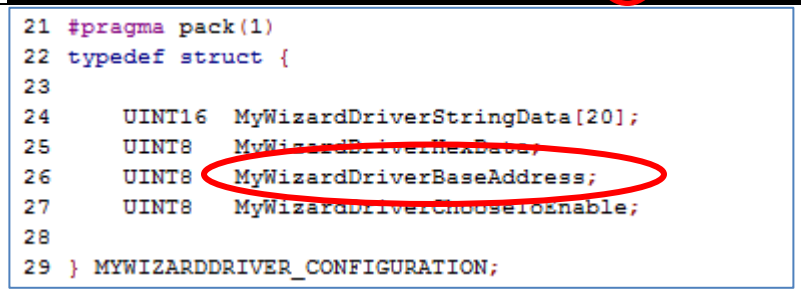
Figure 5 My Wizard Driver with a pop-up box

Step	Action
	Background Information (not a step) The VFR term "oneof" will declare a pop-up menu. The user then selects one field that will dictate the value stored in the NVRAM variable. Looking at Figure 6 above, there are three values: **Value** / **Display** / **String token** **0** / 500 Hex / STR_ONE_OF_TEXT3 **1** / 480 Hex / STR_ONE_OF_TEXT2 **2** / 400 Hex / STR_ONE_OF_TEXT1 Add code to give your driver menu a pop-up menu item by defining a "oneof" item. Also, if the device is "disabled", then use the VFR term "grayoutif" statement so that the pop-up menu is not accessible and cannot be changed. The browser engine will use the configuration variable MWD_IfrNVData.MyWizardDriverChooseToEnable with a value of 0x0 to determine if the device is enabled or disabled
1	**Update** the MyWizardDriver.vfr file
2	**Add** the following code before the "resetbutton" statement (approximately line 53)

Step	Action
	`//` `// Define oneof (EFI_IFR_ONE_OF)` `//` `grayoutif ideqval MWD_IfrNVData.MyWizardDriverChooseToEnable == 0x0;` `oneof name = MyOneOf2, // Define reference name for Question` `varid = MWD_IfrNVData.MyWizardDriverBaseAddress,` `// Use "DataStructure.Member" to reference Buffer Storage` `prompt = STRING_TOKEN(STR_ONE_OF_PROMPT),` `help = STRING_TOKEN(STR_ONE_OF_HELP),` `//` `// Define an option (EFI_IFR_ONE_OF_OPTION)` `//` `option text = STRING_TOKEN(STR_ONE_OF_TEXT3), value = 0x0, flags = 0;` `option text = STRING_TOKEN(STR_ONE_OF_TEXT2), value = 0x1, flags = 0;` `//` `// DEFAULT indicate this option will be marked with` `// EFI_IFR_OPTION_DEFAULT` `//` `option text = STRING_TOKEN(STR_ONE_OF_TEXT1), value = 0x2, flags = DEFAULT;` `endoneof;` `endif;`

Step	Action
	51 default = 1, 52 endcheckbox; 53 // 54 // Define oneof (EFI_IFR_ONE_OF) 55 // 56 grayoutif ideqval MWD_IfrNVData.MyWizardDriverChooseToEnable == 0x0; 57 58 oneof name = MyOneOf2, // Define referenc 59 varid = MWD_IfrNVData.MyWizardDriverBaseAddress, 60 // Use "DataStructure.Member" to reference Buffer Storage 61 prompt = STRING_TOKEN(STR_ONE_OF_PROMPT), 62 help = STRING_TOKEN(STR_ONE_OF_HELP), 63 // 64 // Define an option (EFI_IFR_ONE_OF_OPTION) 65 // 66 option text = STRING_TOKEN(STR_ONE_OF_TEXT3), value = 0x0, flags = 0; 67 option text = STRING_TOKEN(STR_ONE_OF_TEXT2), value = 0x1, flags = 0; 68 // 69 // DEFAULT indicate this option will be marked with 70 // EFI_IFR_OPTION_DEFAULT 71 // 72 option text = STRING_TOKEN(STR_ONE_OF_TEXT1), value = 0x2, 73 flags = DEFAULT; 74 endoneof; 75 endif; 76 77 78 resetbutton 79 defaultstore = MyStandardDefault,
3	**Save** the MyWizardDriver.vfr file
4	**Update** the MyWizardDriver.uni file
5	**Add** the following code to the end of the file (as shown below): **#string STR_ONE_OF_PROMPT #language en "Select Base Address"** **#string STR_ONE_OF_HELP #language en "Select a Base address of 400, 480 or 500 Hex. Values 0,1 or 2(default) is stored in the NVRAM Data"** **#string STR_ONE_OF_TEXT1 #language en "400 Hex"** **#string STR_ONE_OF_TEXT2 #language en "480 Hex"** **#string STR_ONE_OF_TEXT3 #language en "500 Hex"** 33 #string STR_STANDARD_DEFAULT_HELP #language en "This will reset all the Questions to their 34 35 #string STR_ONE_OF_PROMPT #language en "Select Base Address" 36 37 #string STR_ONE_OF_HELP #language en "Select a Base address of 400, 480 or 500 H 38 39 #string STR_ONE_OF_TEXT1 #language en "400 Hex" 40 41 #string STR_ONE_OF_TEXT2 #language en "480 Hex" 42 43 #string STR_ONE_OF_TEXT3 #language en "500 Hex"
6	**Save** MyWizardDriver.uni
7	In the Visual Studio Command Prompt, **type** build
8	**Press** "Enter"

Step	Action
9	**Type** `build run`
10	**Press** "Enter"
11	**Type** `exit`
12	**Press** "Enter"
13	Now at the setup front page menu press the **down arrow** to **"Device Manager"**
14	**Press** "Enter"
15	Inside the Device Manager menu press the down arrow to **"My Wizard Driver Sample Formset"**
16	Down Arrow to "Select Base Address" 
17	**Press** "Enter" **Notice** the Pop up menu
18	**Select** "480 Hex" 
19	**Press** "Enter"
20	**Observe:** Notice the **"Configuration changed"** message at the bottom

Step	Action
21	Test the **"grayoutif"** by selecting "Enable My XYZ Device" then press the "Space" bar to toggle off or "Disabled". 
22	**Notice** the "Select Base Address" is now grayed out and not selectable
23	**Press** "Space" again to Enable
24	**To Exit Press** "Escape" then "Y" or "F10" then "Escape"
25	**To Exit the "Device Manager" Page: Press** "Escape"
26	**Press Up Arrow to "**Continue**"**
27	**At the Shell Prompt type: `dmpstore -all`**
28	
28	 File MyWizardDriverNVDataStruc.h By updating MyWizardDriverNVDataStruc.h, our data structure stored in NVRAM is named MWD_IfrNVData of type MYWIZARDDRIVER_CONFIGURATION. Notice that the base address byte is the next to the last byte in the data structure **`MWD_IfrNVData.MyWizardDriverBaseAddress`** where 02 == 400H, 01 == 480H, 00 == 500H **Notice** the NVRAM Variable with the value of 480H will have a true value of 01.

Step	Action
29	**Type "reset"** at the Shell prompt
30	**Press** "Enter" to return to the Visual Studio Command Prompt 

For any build issues copy the solution files from C:\Fw\LabSolutions\Lesson9.5
NOTE: Del Directory C:\fw\edk2\Build\NT32\DEBUG_VS2010x86\IA32\MyWizardDriver before the Build command to build the MyWizardDriver Clean.

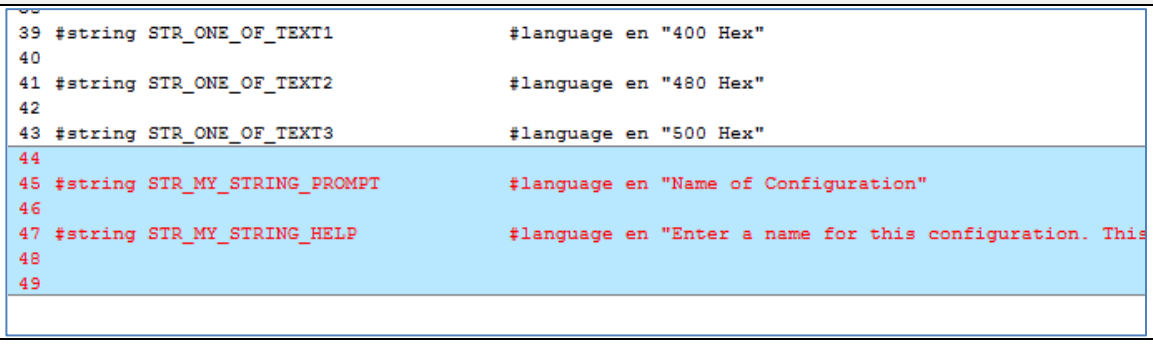
9.6 Updating the Menu: Creating a String to Name a Saved Configuration

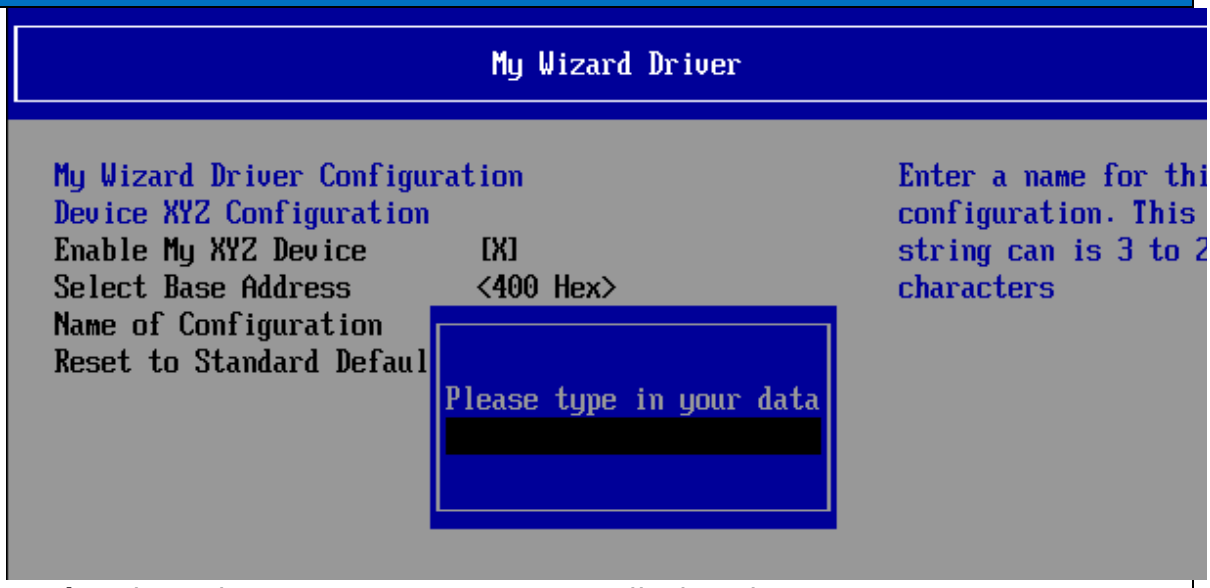
In this lab, you'll create a string to name a saved configuration that will be stored into the NVRAM variable space. This lab uses the VFR term "`string`" to prompt the user to enter a string value. The VFR can determine the minimum and maximum number of characters of the string length with the terms "`minsize`" and "`maxsize`". Since there is also an enable/disable switch, the VFR can use the "`grayoutif`" term again to allow or disallow changes to this field.

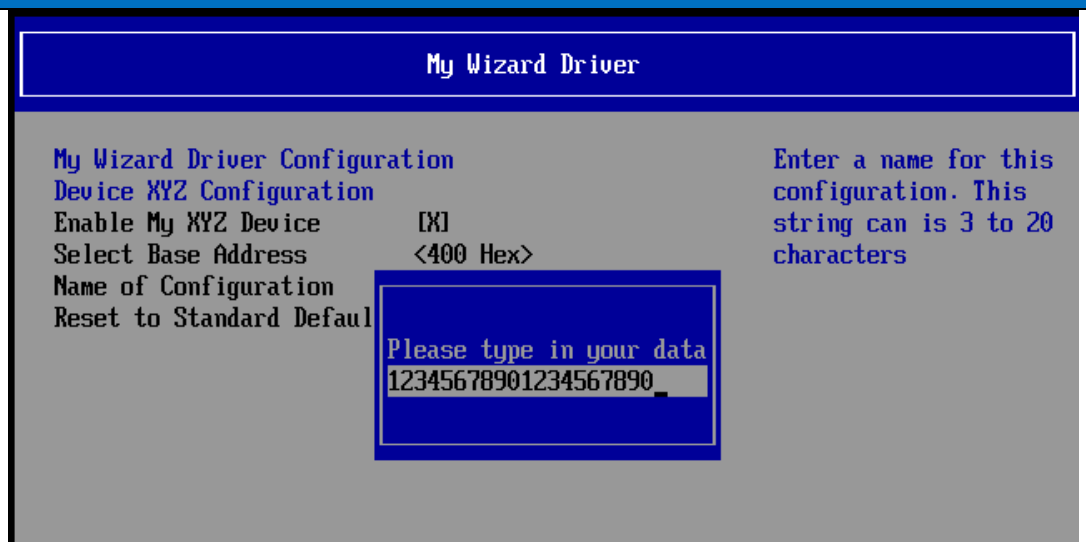
Figure 6 : Menu with a string item

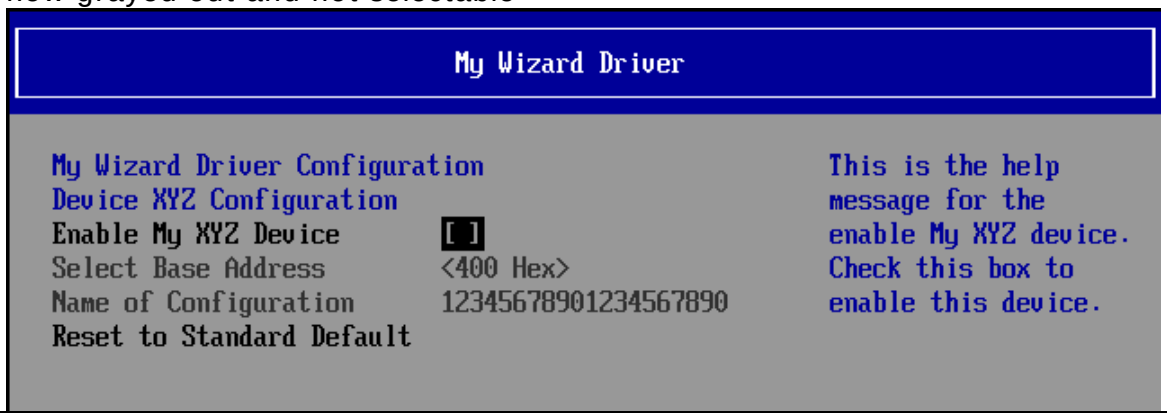
Step	Action
1	**Update** the MyWizardDriver.vfr file
2	**Add** the following code to the location at approx. line 77 and before the "resetbutton" item (as shown below):

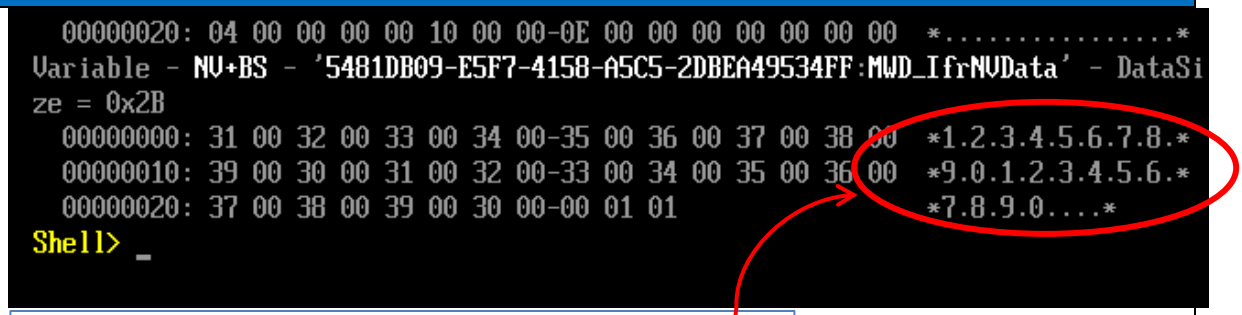
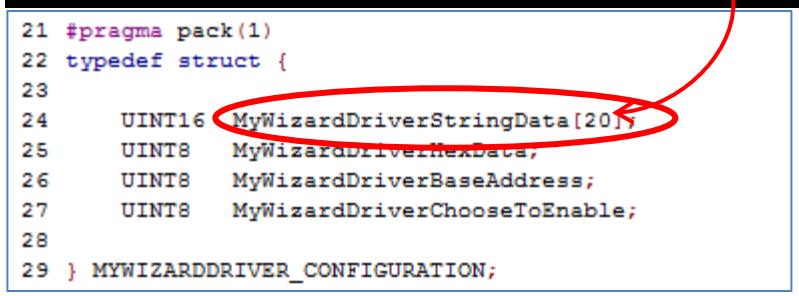
Step	Action
	// // Define a string (EFI_IFR_STRING) to name the configuration in the // NVRAM variable // grayoutif ideqval MWD_IfrNVData.MyWizardDriverChooseToEnable == 0x0; string varid = MWD_IfrNVData.MyWizardDriverStringData, prompt = STRING_TOKEN(STR_MY_STRING_PROMPT), help = STRING_TOKEN(STR_MY_STRING_HELP), minsize = 3, maxsize = 20, endstring; endif;
	74 endoneof; 75 endif; 76 77 78 // 79 // Define a string (EFI_IFR_STRING)to name the configuration in the 80 // NVRAM variable 81 // 82 83 // 84 grayoutif ideqval MWD_IfrNVData.MyWizardDriverChooseToEnable == 0x0; 85 86 string varid = MWD_IfrNVData.MyWizardDriverStringData, 87 prompt = STRING_TOKEN(STR_MY_STRING_PROMPT), 88 help = STRING_TOKEN(STR_MY_STRING_HELP), 89 minsize = 3, 90 maxsize = 20, 91 92 endstring; 93 endif; 94 95 resetbutton 96 defaultstore = MyStandardDefault, 97 prompt = STRING_TOKEN(STR_STANDARD_DEFAULT_PROMPT_RESET), 98 help = STRING_TOKEN(STR_STANDARD_DEFAULT_HELP), 99 endresetbutton; 100
3	**Save** MyWizardDriver.vfr
4	**Update** MyWizardDriver.uni
5	**Add** the following code to the bottom of the file:

Step	Action
	`#string STR_MY_STRING_PROMPT #language en "Name of Configuration"` `#string STR_MY_STRING_HELP #language en "Enter a name for this configuration. This string can is 3 to 20 characters"`
	
6	**Save** MyWizardDriver.uni
7	In the Visual Studio Command Prompt, **type** `build`
8	**Press** "Enter"
9	**Type** `build run`
10	**Press** "Enter"
11	**Type** `exit`
12	**Press** "Enter"
13	Now at the setup front page menu, **select "Device Manager"**
14	**Press** "Enter"
15	Inside the Device Manager menu, **select** "My Wizard Driver Sample Formset"
16	**Select** "Name of Configuration"
17	**Press** "Enter"

Step	Action
18	 **Notice** the string text pop up menu gets displayed
19	**Test** the "`minsize`" by only **typing** your choice of a two character string.
20	**Press** "Enter"  **Notice** the an error message validating that you need to enter three or more characters
21	**Press** "Enter" to clear the message
22	**Press** "Enter" again to re-enter pop up menu

Step	Action
23	 **Test** by **typing** more than "`maxsize`" of 20 characters **Notice** that the Browser only allows the maximum number of 20 characters to be entered with a forced stop. There is no error message but no more characters are allowed to be typed into the pop up menu.
24	**Press** "Enter"  **Notice** that the "**`Configuration changed`**" message is displayed
25	**Test** the `grayoutif` by **selecting** "Enable My XYZ Device"

Step	Action
26	Press the “Spacebar” to toggle off/disable **Notice** that the “Select Base Address” and “Name of Configruation” fields are now grayed out and not selectable 
27	**Press** “Space” again to Enable
28	**Press** “F10” to save
29	**Press** “Escape” to exit
30	**Press** “Escape” to exit the “Device Manager”
31	**Select “**Continue**”**
32	**Press** “Enter”
33	At the Shell Prompt, **type dmpstore -all** **Notice** the unicode string “12345678901234567890” is now stored because you entered those characters in the HII form menu. This is because the file WizardDriverNVDataStruc.h has the data structure stored in NVRAM with the GUID define name MWD_IfrNVData of type MYWIZARDDRIVER_CONFIGURATION. Notice that string data is the first 20 bytes in the data structure MWD_IfrNVData.MyWizardDriverStringData

Step	Action
34	 
35	**Type "reset"** at the Shell prompt
36	**Press** "Enter" to return to the Visual Studio Command Prompt 

For any build issues copy the solution files from C:\Fw\LabSolutions\Lesson9.6
NOTE: Del Directory C:\fw\edk2\Build\NT32\DEBUG_VS2010x86\IA32\MyWizardDriver before the Build command to build the MyWizardDriver Clean.

9.7 Updating the Menu: Numeric Entry

In this lab, you'll learn how to add a numeric entry to your driver menu. This lab uses the VFR term "`numeric`" that prompts the user to enter a free-form numeric value. The VFR determines the minimum and maximum values with the terms "`minimum`" and "`maximum`". Since there is also an enable/disable switch, the VFR uses the "`suppressif`" term to display or hide this field when disabled. Also this field displays as decimal (default) or hexadecimal with the "`flags`" switch.

Figure 7 : Menu with Numeric item entry

Step	Action
1	**Update** the MyWizardDriver.vfr file
2	**Add** the following code in the location shown below at approx. Line 90 and before the "resetbutton" item:

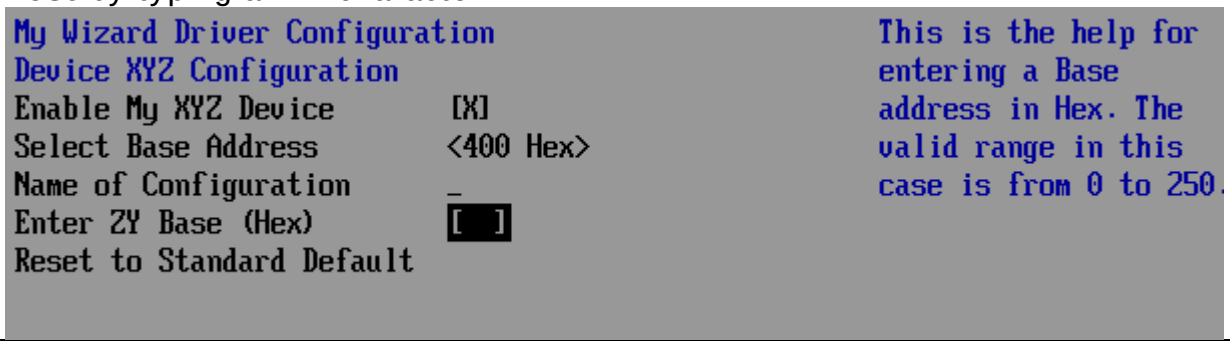
```
//
// Define a numeric free form menu item
//
  suppressif  ideqval MWD_IfrNVData.MyWizardDriverChooseToEnable == 0x0;
   numeric varid   = MWD_IfrNVData.MyWizardDriverHexData,
            prompt  = STRING_TOKEN(STR_DATA_HEX_PROMPT),
            help    = STRING_TOKEN(STR_NUMERIC_HELP),
            flags   = DISPLAY_UINT_HEX ,     // Display in HEX format (if not
specified, default is in decimal format)
            minimum = 0,
            maximum = 250,
          default = 0x22, defaultstore = MyStandardDefault,

    endnumeric;
  endif;
```

```
87      endstring;
88    endif;
89
90  //
91  // Define a numeric free form menu item
92  //
93    suppressif  ideqval MWD_IfrNVData.MyWizardDriverChooseToEnable == 0x0;
94     numeric varid   = MWD_IfrNVData.MyWizardDriverHexData,
95              prompt  = STRING_TOKEN(STR_DATA_HEX_PROMPT),
96              help    = STRING_TOKEN(STR_NUMERIC_HELP),
97              flags   = DISPLAY_UINT_HEX ,     // Display in HEX format (if
98              minimum = 0,
99              maximum = 250,
100             default = 0x22, defaultstore = MyStandardDefault,
101
102    endnumeric;
103   endif;
104
105
106     resetbutton
107       defaultstore = MyStandardDefault,
108       prompt   = STRING_TOKEN(STR_STANDARD_DEFAULT_PROMPT_RESET),
```

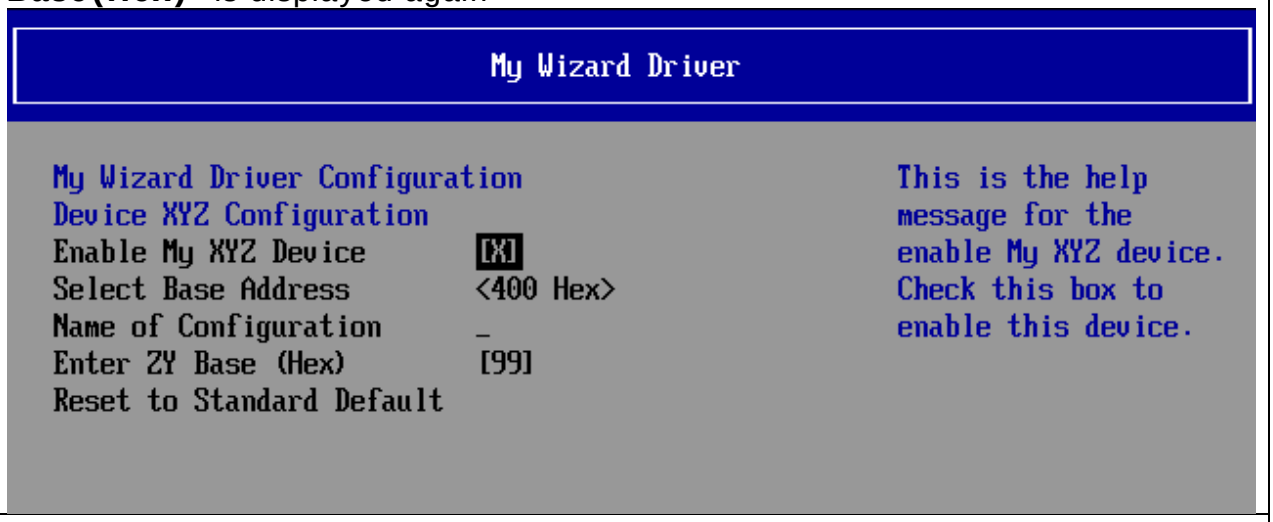
3	**Save** MyWizardDriver.vfr
4	**Update** the MyWizardDriver.uni file
5	**Add** the following code to the bottom of the file:

```
#string STR_DATA_HEX_PROMPT             #language en "Enter ZY Base
(Hex)"

#string STR_NUMERIC_HELP                #language en "This is the
help for entering a Base address in Hex. The valid range in this
case is from 0 to 250."
```

6	**Save** MyWizardDriver.uni
7	In the Visual Studio Command Prompt, **type** build

8	**Press** "Enter"
9	**Type** `build run`
10	**Press** "Enter"
11	**Type** `exit`
12	**Press** "Enter"
13	Now at the setup front page menu, **select** "Device Manager"
14	**Press** "Enter"
15	Inside the Device Manager menu, **select** "My Wizard Driver Sample Formset" **Notice** the value for "Enter ZY Base(Hex)" is 022. Hex is the default because of the VFR field "`default = 0x22`"
16	**Select** "Enter ZY Base(Hex)"
17	**Press** "Enter"
18	**Test** by typing a "M" character 

18	**Notice** that only Numeric characters are allowed and also only values 00 to 0FA Hex. When values outside the range or none numeric characters are entered the red **"!!"** sting is displayed at the bottom of the menu. The string "!!" is part of the Browser engine : MdeModulePkg\Universal\SetupBrowserDxe\SetupBrowserStr.uni #string INPUT_ERROR_MESSAGE #language en-US "!!" 
19	**Press** "Enter" again
20	**Test** by typing a value of '99' Hex
21	**Press** "Enter" **Notice** that the "**Configuration changed**" message is displayed
22	**Test the "surpressif"** by pressing the "spacebar" to "Enable My XYZ Device" then press the "Space" bar to toggle off or "Disabled".

23	**Notice the "Select Base Address" and "Name of Configuration"** fields are now grayed out and not selectable and the "**Enter ZY Base(Hex)**" does not appear at all. 
24	**Press** "Space bar" again to "Enable My XYZ Device" and the "**Enter ZY Base(Hex)**" is displayed again 
25	**Press** "F10" then "Escape" to exit
26	**Press** "Escape" to exit the "Device Manager"
27	**Select "**Continue**"**
28	**Press** "Enter"
29	At the Shell Prompt, **type** `dmpstore –all`

30

```
Variable - NV+BS - '5481DB09-E5F7-4158-A5C5-2DBEA49534FF:MWD_IfrNVData' - DataSi
ze = 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 00-99 02 01                 *...........*
Shell> _
```

```
21 #pragma pack(1)
22 typedef struct {
23
24     UINT16  MyWizardDriverStringData[20];
25     UINT8   MyWizardDriverHexData;
26     UINT8   MyWizardDriverBaseAddress;
27     UINT8   MyWizardDriverChooseToEnable;
28
29 } MYWIZARDDRIVER_CONFIGURATION;
```

Notice by modifying MyWizardDriverNVDataStruc.h our data structure stored in NVRAM is named **MWD_IfrNVData** of type MYWIZARDDRIVER_CONFIGURATION.

Notice that hex data is after the string data at the 21st byte in the data structure **MWD_IfrNVData.MyWizardDriverHexData**

31 **Type "reset"** at the Shell prompt

32 **Press** "Enter" to return to the Visual Studio Command Prompt

```
C:\FW\edk2>
```

For any build issues copy the solution files from C:\Fw\LabSolutions\Lesson9.7
NOTE: Del Directory C:\fw\edk2\Build\NT32\DEBUG_VS2010x86\IA32\MyWizardDriver before the Build command to build the MyWizardDriver Clean.

9.8 Updating your Driver for Interactive Call Backs

In this lab, you'll update your driver for interactive call backs. Call backs are a way to communicate changes the user is making in "real time" where your driver needs to intervene as the changes are made and before the user exits the current menu being displayed. These would be exception cases that the driver could interrupt the normal browser engine process.

To add call backs, the file HiiConfigAccess.c of your driver will be updated in the function MyWizardDriverHiiConfigAccessCallback. This function is called whenever any VFR items have a flag for INTERACTIVE set. So far, the previous labs did not have any call back items.

We can see this because there was a "Debug" call made in the MyWizardDriverHiiConfigAccessCallback function that never gets called:

HiiConfigAccess.c (line 331)

```
DEBUG ((DEBUG_INFO, "\n:: START Call back ,Question ID=0x%08x Type=0x%04x
Action=0x%04x", QuestionId, Type, Action));
```

9.8.1 *Add the Case statements to the Call back routine*

Step	Action
1	**Update** the HiiConfigAccess.c file
2	**Add** the following code before `return status;` to include a "case" statement in the call back routine for the "action" passed. Add the following code at approx. line 343 before: **`return status`**;

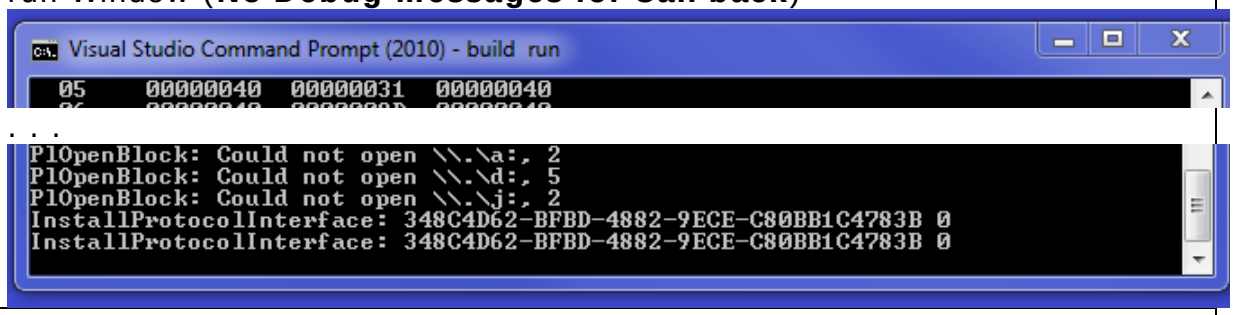
Step	Action

```
switch (Action) { // Start switch and passed param Action
case EFI_BROWSER_ACTION_FORM_OPEN:  // 3
  {
  }
  break;

case EFI_BROWSER_ACTION_FORM_CLOSE: // 4
  {
  }
  break;

case EFI_BROWSER_ACTION_RETRIEVE: // 2
  {
  }
  break;

case EFI_BROWSER_ACTION_DEFAULT_STANDARD: // 0x1000
  {
  }
  break;

case EFI_BROWSER_ACTION_DEFAULT_MANUFACTURING: // 0x1001
  {
  }
  break;
case EFI_BROWSER_ACTION_CHANGING:  // 0
  {
  }
  break;

case EFI_BROWSER_ACTION_CHANGED:  // 1
  {
  }
  break;

default:
  Status = EFI_UNSUPPORTED;
  break;
}  // end switch case on Action
```

Step	Action

```
340   FormId = 0;
341   Status = EFI_SUCCESS;
342   PrivateData = MYWIZARDDRIVER_DEV_FROM_THIS (This);
343
344
345   switch (Action) { // Start switch and passed param Action
346   case EFI_BROWSER_ACTION_FORM_OPEN:  // 3
347     {
348         }
349     break;
350
351   case EFI_BROWSER_ACTION_FORM_CLOSE: // 4
352     {
353         }
354         break;
355
356   case EFI_BROWSER_ACTION_RETRIEVE: // 2
357     {
358         }
359         break;
360
361   case EFI_BROWSER_ACTION_DEFAULT_STANDARD: // 0x1000
362     {
363         }
364         break;
365
366   case EFI_BROWSER_ACTION_DEFAULT_MANUFACTURING: // 0x1001
367     {
368         }
369         break;
370
371   case EFI_BROWSER_ACTION_CHANGING:  // 0
372     {
373         }
374     break;
375
376   case EFI_BROWSER_ACTION_CHANGED:  // 1
377     {
378         }
379     break;
380
381   default:
382     Status = EFI_UNSUPPORTED;
383     break;
384   }  // end switch case on Action
385
386   return Status;
387
388 //  return EFI_UNSUPPORTED;
389 }
```

3	**Save** HiiConfigAccess.c
4	In the Visual Studio Command Prompt, **type** `build`

Step	Action
5	**Press** "Enter"
6	**Type** `build run`
7	**Press** "Enter"
8	**Type** `exit`
9	**Press** "Enter"
10	Now at the setup front page menu, **select** "Device Manager"
11	**Press** "Enter"
12	Inside the Device Manager menu, **select** "My Wizard Driver Sample Formset"
13	**Press** "Enter"
14	**Notice** the debug messages in the Visual Studio Command Prompt – build run Window (**No Debug messages for Call back**) 
15	**Press** "Escape" to exit
16	**Press** "Escape" to exit the "Device Manager"
17	**Select "**Continue**"**
18	**Press** "Enter"
19	**Type** "`reset`" at the Shell prompt
20	**Press** "Enter" to return to the Visual Studio Command Prompt 

9.8.2 *Update the Menu for Interactive items*

1	**Update** the MyWizardDriver.vfr file
2	Now, you'll add the flag characteristic `INTERACTIVE` to the string item's flags by using keyword `INTERACTIVE` and `questionid`. **Add** the following code in the location shown below: Approx. line 83 and line 86
	`questionid = 0x1001,`
	`flags = INTERACTIVE,`

```
76
77  //
78  // Define a string (EFI_IFR_STRING) to name the configuration in the
79  // NVRAM variable
80      //
81    grayoutif  ideqval MWD_IfrNVData.MyWizardDriverChooseToEnable == 0x0;
82      string    varid     = MWD_IfrNVData.MyWizardDriverStringData,
83             questionid   = 0x1001,
84                prompt    = STRING_TOKEN(STR_MY_STRING_PROMPT),
85                help      = STRING_TOKEN(STR_MY_STRING_HELP),
86                flags     = INTERACTIVE,
87                minsize   = 3,
88                maxsize   = 20,
89      endstring;
90    endif;
91
```

3 Include the numeric item by adding the following code in the location shown below, Approx. line 97 and line 100

```
questionid = 0x1111,
| INTERACTIVE
```

```
92  //
93  // Define a numeric free form menu item
94  //
95    suppressif  ideqval MWD_IfrNVData.MyWizardDriverChooseToEnable == 0x0;
96     numeric varid    = MWD_IfrNVData.MyWizardDriverHexData,
97             questionid  = 0x1111,
98              prompt   = STRING_TOKEN(STR_DATA_HEX_PROMPT),
99              help     = STRING_TOKEN(STR_NUMERIC_HELP),
100             flags    = DISPLAY_UINT_HEX | INTERACTIVE,     // Display in HEX
101             minimum = 0,
102             maximum = 250,
103             default = 0x22, defaultstore = MyStandardDefault,
104
105    endnumeric;
106  endif;
```

4 **Save** MyWizardDriver.vfr

5 In the Visual Studio Command Prompt, **type** `build`

6 **Press** "Enter"

7 **Type** `build run`

8 **Press** "Enter"

9 **Type** `exit`

10 **Press** "Enter"

11 Now at the setup front page menu, **select** "Device Manager"

12 **Press** "Enter"

13 Inside the Device Manager menu, **select** "My Wizard Driver Sample Formset"

14 **Press** "Enter"

15 Take a moment and **review** the Visual Studio build run command prompt window

16 In the NT32 emulation window, **click** on "Name of Configuration" and "Enter ZY Base(Hex)"

17 **Notice** the following in the Visual Studio Command Prompt window:
Every time the browser does anything with the interactive labeled fields there is a call made to your driver's call back function. We can determine which item by the quetionid and what action by the Action passed to your call back function. Your call back function can then add code to special case when these transitions occur.

Entering Form

```
:: START Call back ,Question ID=0x00001001 Type=0x0007 Action=0x0003
:: START Call back ,Question ID=0x00001111 Type=0x0000 Action=0x0003
:: START Call back ,Question ID=0x00001001 Type=0x0007 Action=0x0002
```

Changing a Value for Question ID 0x1111

```
:: START Call back ,Question ID=0x00001001 Type=0x0007 Action=0x0003
:: START Call back ,Question ID=0x00001111 Type=0x0000 Action=0x0003
:: START Call back ,Question ID=0x00001001 Type=0x0007 Action=0x0002
:: START Call back ,Question ID=0x00001111 Type=0x0000 Action=0x0002
:: START Call back ,Question ID=0x00001111 Type=0x0000 Action=0x0000
:: START Call back ,Question ID=0x00001111 Type=0x0000 Action=0x0001
:: START Call back ,Question ID=0x00001001 Type=0x0007 Action=0x0002
```

Changing a Value for Question ID 0x1001

```
:: START Call back ,Question ID=0x00001001 Type=0x0007 Action=0x0003
:: START Call back ,Question ID=0x00001111 Type=0x0000 Action=0x0003
:: START Call back ,Question ID=0x00001001 Type=0x0007 Action=0x0002
:: START Call back ,Question ID=0x00001111 Type=0x0000 Action=0x0002
:: START Call back ,Question ID=0x00001111 Type=0x0000 Action=0x0000
:: START Call back ,Question ID=0x00001111 Type=0x0000 Action=0x0001
:: START Call back ,Question ID=0x00001001 Type=0x0007 Action=0x0002
:: START Call back ,Question ID=0x00001001 Type=0x0007 Action=0x0001
:: START Call back ,Question ID=0x00001001 Type=0x0007 Action=0x0002

:: START Call back ,Question ID=0x00001001 Type=0x0007 Action=0x0003
:: START Call back ,Question ID=0x00001111 Type=0x0000 Action=0x0003
:: START Call back ,Question ID=0x00001001 Type=0x0007 Action=0x0002
:: START Call back ,Question ID=0x00001111 Type=0x0000 Action=0x0002
:: START Call back ,Question ID=0x00001111 Type=0x0000 Action=0x0000
:: START Call back ,Question ID=0x00001111 Type=0x0000 Action=0x0001
:: START Call back ,Question ID=0x00001001 Type=0x0007 Action=0x0002
:: START Call back ,Question ID=0x00001111 Type=0x0000 Action=0x0002
:: START Call back ,Question ID=0x00001001 Type=0x0007 Action=0x0000
:: START Call back ,Question ID=0x00001001 Type=0x0007 Action=0x0002
:: START Call back ,Question ID=0x00001111 Type=0x0000 Action=0x0002
:: ROUTE CONFIG Saving the configuration to NVRAM

:: START Call back ,Question ID=0x00001001 Type=0x0007 Action=0x0003
:: START Call back ,Question ID=0x00001111 Type=0x0000 Action=0x0003
:: START Call back ,Question ID=0x00001001 Type=0x0007 Action=0x0002
:: START Call back ,Question ID=0x00001111 Type=0x0000 Action=0x0002
:: START Call back ,Question ID=0x00001111 Type=0x0000 Action=0x0000
:: START Call back ,Question ID=0x00001111 Type=0x0000 Action=0x0001
:: START Call back ,Question ID=0x00001001 Type=0x0007 Action=0x0002
:: START Call back ,Question ID=0x00001111 Type=0x0000 Action=0x0002
:: START Call back ,Question ID=0x00001001 Type=0x0007 Action=0x0000
:: START Call back ,Question ID=0x00001001 Type=0x0007 Action=0x0001
:: START Call back ,Question ID=0x00001001 Type=0x0007 Action=0x0002
:: START Call back ,Question ID=0x00001111 Type=0x0000 Action=0x0002
:: ROUTE CONFIG Saving the configuration to NVRAM

:: START Call back ,Question ID=0x00001001 Type=0x0007 Action=0x0004
:: START Call back ,Question ID=0x00001111 Type=0x0000 Action=0x0004Ins
colInterface: 348C4D62-BFBD-4882-9ECE-C80BB1C4783B 0
```

18	**Press** "Escape" to exit
19	**Press** "Escape" to exit the "Device Manager"
20	**Select "**Continue**"**
21	**Press** "Enter"
22	**Type** "`reset`" at the Shell prompt
23	**Press** "Enter" to return to the Visual Studio Command Prompt 

For any build issues copy the solution files from C:\Fw\LabSolutions\Lesson9.8
NOTE: Del Directory C:\fw\edk2\Build\NT32\DEBUG_VS2010x86\IA32\MyWizardDriver before the Build command to build the MyWizardDriver Clean.

9.9 Add code to your driver when Call Back events occur for Interactive Items

In this lab, you'll update your driver to print debug statements when the Hii browser engine calls back into your call back function. Every time the browser does anything with the interactive labeled fields there is a call made to your driver's call back function. We can determine the item by the quetionid and what action based on the action passed to your call back function. Your call back function can then add code to special case when these transitions occur.

For this lab we will simply add Debug print statements. However, the use of adding call backs to a driver's HII functions adds the capability of providing more manageability and flexibility for the interactions between the user, the browser engine, and your driver code. In a real driver firmware situation, it may be desired to implement more complex features and functionality based upon an item changing.

Step	Action
1	**Update** the HiiConfigAccess.c file
2	**Comment out** the DEBUG statement with "//" in the MyWizardDriverHiiConfigAccessCallback call back function approx. line 330: Because this will get called so many times that it will be hard to determine where your code is actually doing something
	//
	326 MYWIZARDDRIVER_DEV *PrivateData; 327 EFI_STATUS Status; 328 EFI_FORM_ID FormId; 329 330 // DEBUG ((DEBUG_INFO, "\n:: START Call back ,Question ID=0 331 332 333 334 if (((Value == NULL) && (Action != EFI_BROWSER_ACTION_FORM_O
3	**Add** a switch case statement of the question ID's to the "Action" switch case of EFI_BROWSER_ACTION_CHANGING in the call back function by **adding** a nested switch case code (as shown below at approx. line 372)

Step	Action

```
            switch (QuestionId) {
                  case 0x1111:
                              DEBUG ((DEBUG_INFO, "\n:: START Call back-
Changing ,Question ID=0x%08x Type=0x%04x Action=0x%04x", QuestionId, Type,
Action));
                        break;
                  case 0x1001:
                              DEBUG ((DEBUG_INFO, "\n:: START Call back-
Changing ,Question ID=0x%08x Type=0x%04x Action=0x%04x", QuestionId, Type,
Action));
                        break;
                  default:
                        Status = EFI_UNSUPPORTED;
                        break;
            }
```

```
369
370   case EFI_BROWSER_ACTION_CHANGING:  // 0
371     {
372                   switch (QuestionId) {
373                          case 0x1111:
374                                      DEBUG ((DEBUG_INFO, "\n:: START Call back- Chan
375                                 break;
376                          case 0x1001:
377                                      DEBUG ((DEBUG_INFO, "\n:: START Call back- Chan
378                                 break;
379                          default:
380                                 Status = EFI_UNSUPPORTED;
381                                 break;
382                   }
383         }
384     break;
385
386   case EFI_BROWSER_ACTION_CHANGED:  // 1
```

4	**Add** another nested switch case statement of the question ID's to the "Action" switch case of EFI_BROWSER_ACTION_CHANGED in the call back function (as show below at approx. line 388):

Step	Action
	switch (QuestionId) { case 0x1111: DEBUG ((DEBUG_INFO, "\n:: START Call back-Changed ,Question ID=0x%08x Type=0x%04x Action=0x%04x", QuestionId, Type, Action)); break; case 0x1001: DEBUG ((DEBUG_INFO, "\n:: START Call back-Changed ,Question ID=0x%08x Type=0x%04x Action=0x%04x", QuestionId, Type, Action)); break; default: Status = EFI_UNSUPPORTED; break; }
	385 386 case EFI_BROWSER_ACTION_CHANGED: // 1 387 { 388 switch (QuestionId) { 389 case 0x1111: 390 DEBUG ((DEBUG_INFO, "\n:: START Call back- Changed 391 break; 392 case 0x1001: 393 DEBUG ((DEBUG_INFO, "\n:: START Call back- Changed 394 break; 395 default: 396 Status = EFI_UNSUPPORTED; 397 break; 398 } 399 } 400 break; 401 402 default: 403 Status = EFI_UNSUPPORTED; 404 break; 405 } // end switch case on Action 406 407 return Status; 408
5	**Save** MyWizardDriver.c
6	In the Visual Studio Command Prompt, **type** build
7	**Press** "Enter"
8	**Type** build run
9	**Type** exit
10	**Press** "Enter"
11	Now at the setup front page menu, **select** "Device Manager"
12	**Press** "Enter"
13	Inside the Device Manager menu, **select** "My Wizard Driver Sample Formset"
14	**Press** "Enter"

Step	Action
15	**Observe** the Visual Studio Command Prompt – build run Window **Test:** changing the **"Name of Configuration"** and the "**Enter ZY Base(Hex)**" fields while observing the Visual Studio Command Prompt – build run Window
16	**Switch back** to Visual Studio and notice the changes that you made.
17	`:: START Call back- Changing ,Question ID=0x00001001 Type=0x0007 Action=0x0000` **Notice:** when changing the "**Name of Configuration**" field
18	`:: START Call back- Changing ,Question ID=0x00001001 Type=0x0007 Action=0x0000` `:: START Call back- Changed ,Question ID=0x00001001 Type=0x0007 Action=0x0001` `:: START Call back- Changing ,Question ID=0x00001111 Type=0x0000 Action=0x0000` **Notice:** when changing the "**Enter ZY Base(Hex)**" field
19	`:: START Call back- Changing ,Question ID=0x00001001 Type=0x0007 Action=0x0000` `:: START Call back- Changed ,Question ID=0x00001001 Type=0x0007 Action=0x0001` `:: START Call back- Changing ,Question ID=0x00001111 Type=0x0000 Action=0x0000` `:: START Call back- Changed ,Question ID=0x00001111 Type=0x0000 Action=0x0001` `:: ROUTE CONFIG Saving the configuration to NVRAM` **Notice:** when Pressing "F10"
20	**Press** "Escape" to exit
21	**Press** "Escape" to exit the "Device Manager"
22	**Select "**Continue**"**
23	**Press** "Enter"
24	**Type** "`reset`" at the Shell prompt
25	**Press** "Enter" to return to the Visual Studio Command Prompt `C:\FW\edk2>`

For any build issues copy the solution files from C:\Fw\LabSolutions\Lesson9.9
NOTE: Del Directory C:\fw\edk2\Build\NT32\DEBUG_VS2010x86\IA32\MyWizardDriver before the Build command to build the MyWizardDriver Clean.

9.10 Adding an Additional Form Page

In this lab, you'll learn how to add another form page to your My Wizard Driver menu by using the "`goto`" VFR term along with the "`form`" and "`formid`" VFR statements. Additionally, use "`surpressif`" or "`grayoutif`" to conditionally allow the user to enter your additional forms.

In addition, this lab will show how the "`time`" and "`date`" VFR terms are used within the VFR language to special case how the browser engine checks the time instead of your driver manually checking (e.g. leap year).

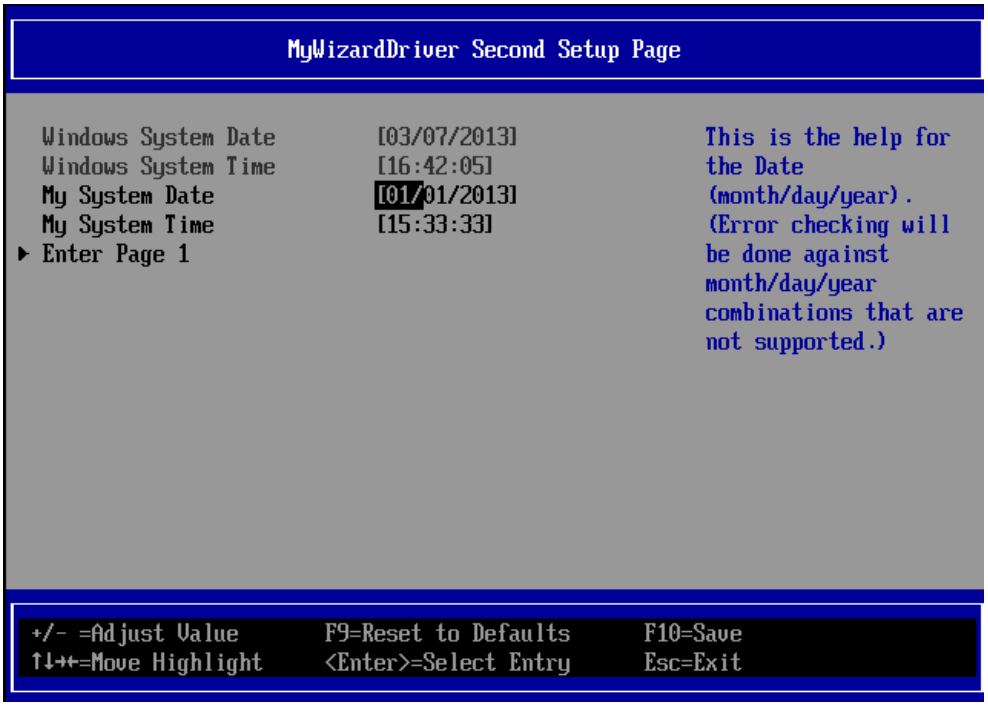
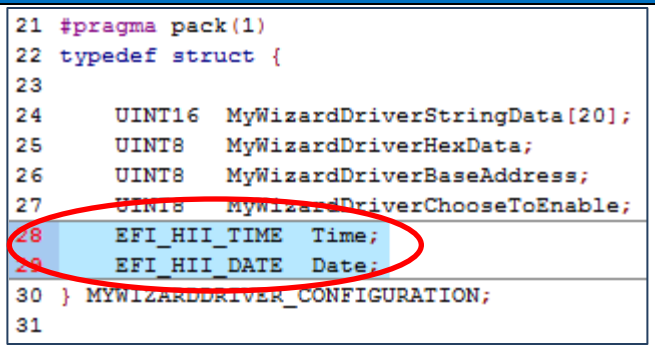


Figure 10: Second setup page

Step	Action
1	**Update** the MyWizardDriverNVDataStruc.h file
2	**Add** the following date and time fields to the configuration `typedef` (to to the location shown below):
	`EFI_HII_TIME Time;` `EFI_HII_DATE Date;`

Step	Action
	

```
21 #pragma pack(1)
22 typedef struct {
23
24     UINT16  MyWizardDriverStringData[20];
25     UINT8   MyWizardDriverHexData;
26     UINT8   MyWizardDriverBaseAddress;
27     UINT8   MyWizardDriverChooseToEnable;
28     EFI_HII_TIME  Time;
29     EFI_HII_DATE  Date;
30 } MYWIZARDDRIVER_CONFIGURATION;
31
```

3	**Save** MyWizardDriverNVDataStruc.h
4	**Update** the MyWizardDriver.uni file
5	**Add** the following code to the end of the file to update the second page's string:

```
#string STR_FORM2_TITLE                 #language en "MyWizardDriver
Second Setup Page"
#string STR_DATE_PROMPT                 #language en "Windows System
Date"
#string STR_DATE_HELP                   #language en "This is the help
for the Date (month/day/year). (Error checking will be done against
month/day/year combinations that are not supported.)"
#string STR_TIME_PROMPT                 #language en "Windows System
Time"
#string STR_TIME_HELP                   #language en "This is the help
for the Time (hour/minute/second)."
#string STR_ERROR_POPUP                 #language en "You typed in the
wrong value!"

#string STR_GOTO_FORM1                  #language en "Enter Page 1"
#string STR_GOTO_FORM2                  #language en "Enter Page 2"
#string STR_GOTO_HELP                   #language en "This is my goto
help"

#string STR_MY_DATE_PROMPT              #language en "My System Date"

#string STR_MY_TIME_PROMPT              #language en "My System Time"
```

6	**Save** MyWizardDriver.uni
7	**Update** the MyWizardDriver.vfr file
8	**Add** the "goto" VFR item to allow browser to ender another form by adding the following code before the "**endform"** at approx. line 114

```
   grayoutif  ideqval MWD_IfrNVData.MyWizardDriverChooseToEnable
== 0x0;
        goto 2,
      prompt = STRING_TOKEN(STR_GOTO_FORM2), //SecondSetupPage
      help   = STRING_TOKEN(STR_GOTO_HELP);
   endif;
```

Step	Action
	108 resetbutton 109 defaultstore = MyStandardDefault, 110 prompt = STRING_TOKEN(STR_STANDARD_DEFAULT_PROMPT_RESET), 111 help = STRING_TOKEN(STR_STANDARD_DEFAULT_HELP), 112 endresetbutton; 113 114 grayoutif ideqval MWD_IfrNVData.MyWizardDriverChooseToEnable == 0x0; 115 goto 2, 116 prompt = STRING_TOKEN(STR_GOTO_FORM2), //SecondSetupPage 117 help = STRING_TOKEN(STR_GOTO_HELP); 118 endif; 119 120 endform; 121
9	**Add** the following code between "**endform**" at approx. line 120 and "**endformset**" (the code continues for three pages in this lab guide):

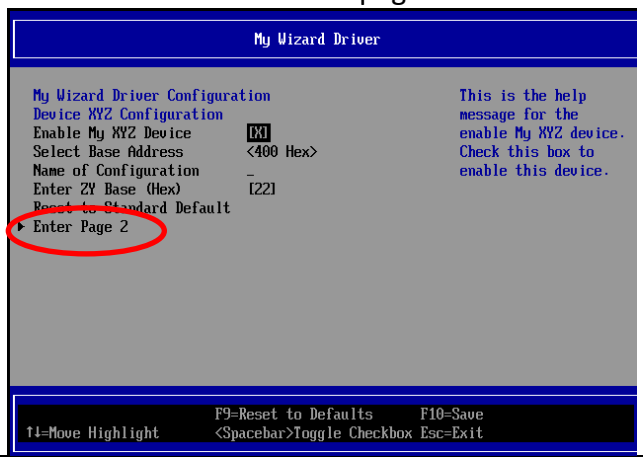
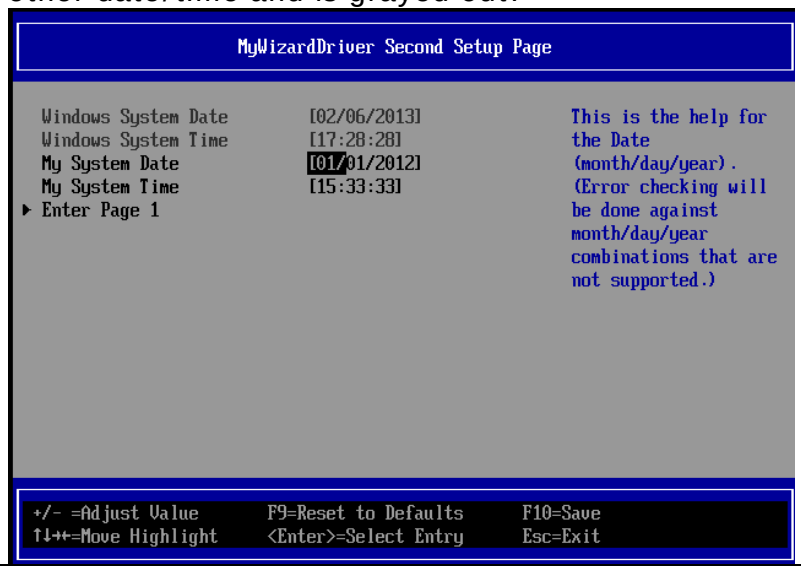
Step	Action

```
    form formid = 2,               // SecondSetupPage,
      title = STRING_TOKEN(STR_FORM2_TITLE);

      grayoutif TRUE; // DATE is the date of the Windows Host so can
not change it.;
      date    year varid  = Date.Year, // Note that it is a member of
NULL,
                //so the RTC will be the system resource to retrieve
and save from
              prompt      = STRING_TOKEN(STR_DATE_PROMPT),
              help        = STRING_TOKEN(STR_DATE_HELP),
              minimum     = 1998,
              maximum     = 2099,
              step        = 1,
              default     = 2010,

              month varid = Date.Month, // Note that it is a member of
NULL,
                 //so the RTC will be the system resource to retrieve
and save from
              prompt      = STRING_TOKEN(STR_DATE_PROMPT),
              help        = STRING_TOKEN(STR_DATE_HELP),
              minimum     = 1,
              maximum     = 12,
              step        = 1,
              default     = 1,

              day varid   = Date.Day, // Note that it is a member of
NULL,
                //so the RTC will be the system resource to retrieve
and save from
              prompt      = STRING_TOKEN(STR_DATE_PROMPT),
              help        = STRING_TOKEN(STR_DATE_HELP),
              minimum     = 1,
              maximum     = 31,
              step        = 0x1,
              default     = 1,
      enddate;
    endif;  //grayoutif TRUE DATE
```

Step	Action

```
    grayoutif TRUE;  // TIME – WINDOWS TIME
      time    hour varid  = Time.Hour, // Note that it is a
member of NULL,
                //so the RTC will be the system resource to
retrieve and save from
              prompt      = STRING_TOKEN(STR_TIME_PROMPT),
              help        = STRING_TOKEN(STR_TIME_HELP),
              minimum     = 0,
              maximum     = 23,
              step        = 1,
              default     = 0,

              minute varid  = Time.Minute, // Note that it is a
member of NULL,
                 //so the RTC will be the system resource to
retrieve and save from
              prompt        = STRING_TOKEN(STR_TIME_PROMPT),
              help          = STRING_TOKEN(STR_TIME_HELP),
              minimum       = 0,
              maximum       = 59,
              step          = 1,
              default       = 0,

              second varid  = Time.Second, // Note that it is a
member of NULL,
                 //so the RTC will be the system resource to
retrieve and save from
              prompt        = STRING_TOKEN(STR_TIME_PROMPT),
              help          = STRING_TOKEN(STR_TIME_HELP),
               minimum       = 0,
              maximum       = 59,
              step          = 1,
              default       = 0,

      endtime;
    endif; //grayoutif TRUE  TIME
```

Step	Action
	`` date // My Wizard Driver Date varid = MWD_IfrNVData.Date , prompt = STRING_TOKEN(STR_MY_DATE_PROMPT), help = STRING_TOKEN(STR_DATE_HELP), flags = STORAGE_NORMAL, default = 2013/01/01, enddate; time // My Wizard Driver Time name = MyTimeMWD, varid = MWD_IfrNVData.Time, prompt = STRING_TOKEN(STR_MY_TIME_PROMPT), help = STRING_TOKEN(STR_TIME_HELP), flags = STORAGE_NORMAL , default = 15:33:33, endtime; goto 1, prompt = STRING_TOKEN(STR_GOTO_FORM1), //MainSetupPage // this too has no end-op and basically it's a jump to a form ONLY help = STRING_TOKEN(STR_GOTO_HELP); endform; ```
10	**Save** MyWizardDriver.vfr
11	In the Visual Studio Command Prompt, **type** `build`
12	**Press** “Enter”
13	**Type** `build run`
14	**Press** “Enter”
15	**Type** `exit`
16	Now at the setup front page menu, **select** “Device Manager”
17	**Press** “Enter”
18	Inside the Device Manager menu, **select** “My Wizard Driver Sample Formset”

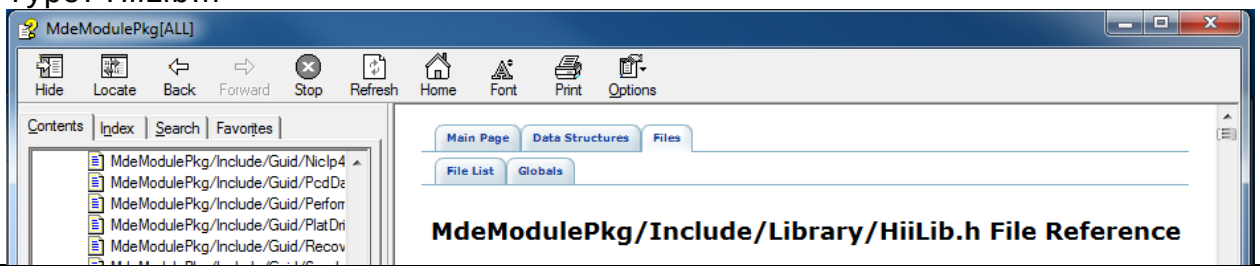
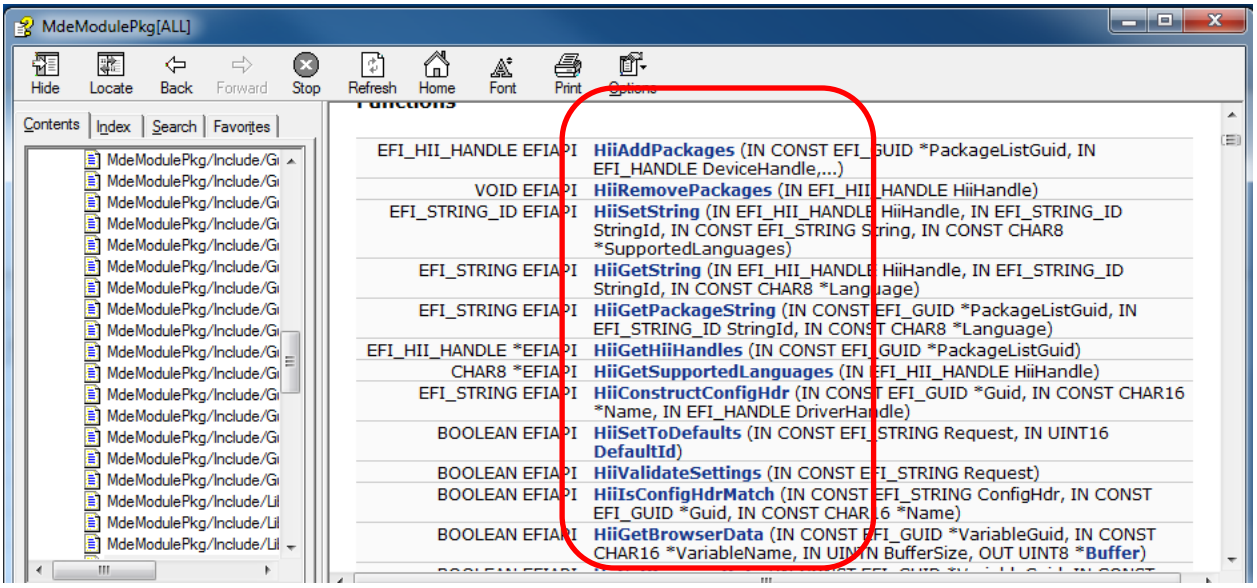
Step	Action
19	**Press** "Enter" **Notice** the "Enter Page 2" option. Without `goto` in the MyWizardDriver.vfr file, you wouldn't be able to access page two. 
20	**Select** "Enter Page 2"
21	**Press** "Enter" **Notice** how the Windows System Date and Time cannot be modified to any other date/time and is grayed out: 
22	**Test** by trying to enter the date 02/30/2013, then try a valid leap year date: 02/29/2012.
23	**Press** "Down Arrow" to return to Page 1
24	Test the "`grayoutif`" by going to "Enable My XYZ Device"
25	Press the "Spacebar" to toggle off/disable

Step	Action
26	**Notice** the "Select Base Address" , "Name of Configruation" and the "Enter Page 2" fields are now grayed out and not selectable  **Note**: If you wanted to hide "**Enter Page 2**", you would simply replace the `grayoutif` code you entered with `suppressif` in the VFR file
27	**Press** "Space bar" again to Enable
28	**Press** "F10" then "Escape" to save and exit
29	**Press** "Escape" to exit "Device Manager"
30	**Select "**Continue**"**
31	**Press** "Enter"
32	**Type** "`reset`" 
33	**Press** "Enter" to return to the Visual Studio Command Prompt 

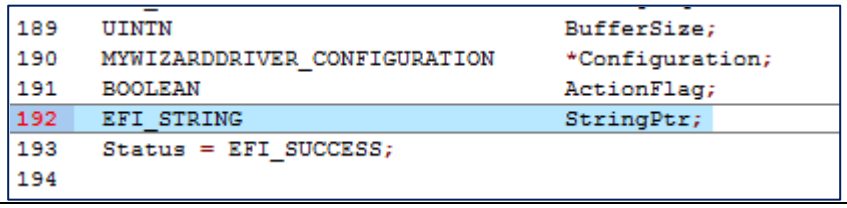
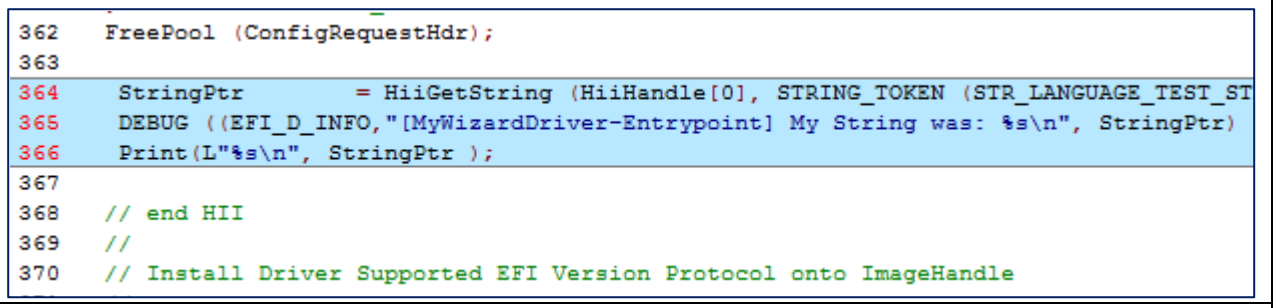
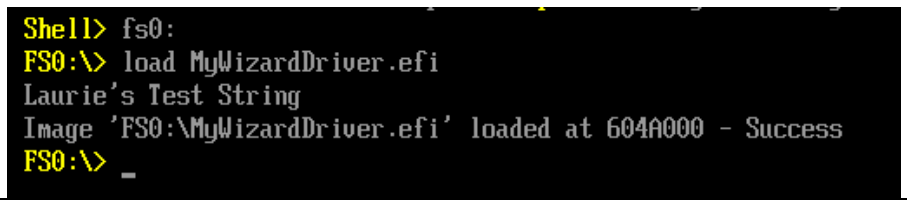
For any build issues copy the solution files from C:\Fw\LabSolutions\Lesson9.10
NOTE: Del Directory C:\fw\edk2\Build\NT32\DEBUG_VS2010x86\IA32\MyWizardDriver before the Build command to build the MyWizardDriver Clean.

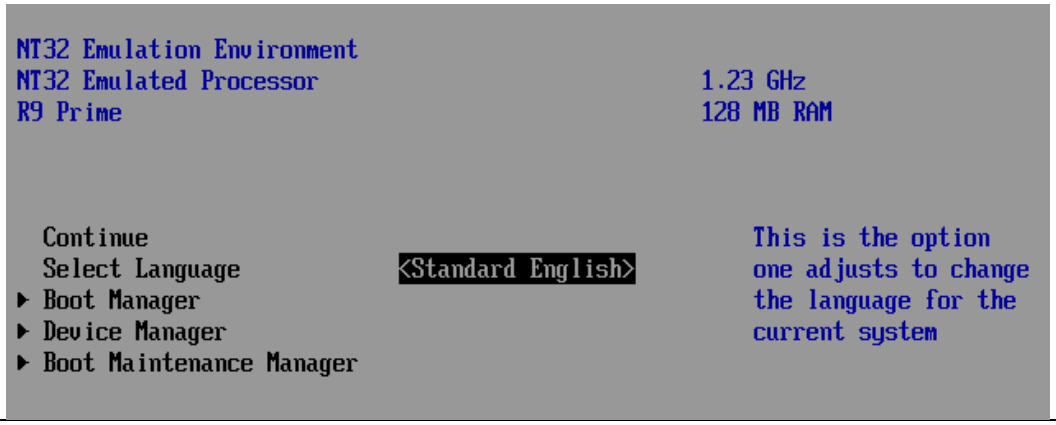
9.11 Adding Communication from Driver to Console through HII

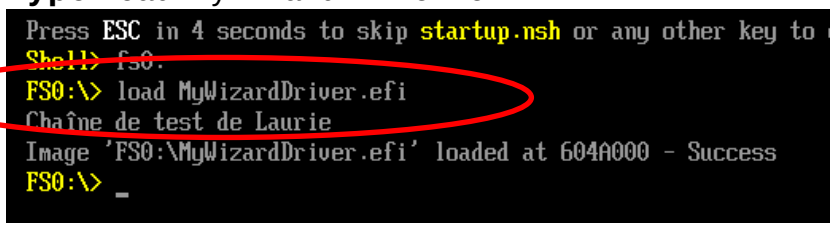
In this lab, you'll add communication from the driver to the console through HII. More specifically, you'll add code to retrieve a string from the HII database and print the string to the console. Then, you'll add the string in French, change the language, and test to ensure the correct language is displayed. The reason the driver should avoid direct string text to the console without the HII support is because there is no localization for text string inside the driver's source code. By using the HII database the strings are tokenized making localization easier.

Step	Action
1	For HII database library functions open the MdeModulePkg .Chm file and search on HiiLib.h functions
2	Select the Index tab
3	Type: HiiLib.h 
4	 ***Note***: Notice the list of Hii function calls available. To get Strings the `HiiGetString` function can be used.
5	**Update** the C:\Fw\edk2\Nt32pkg\Nt32pkg.fdf file

Step	Action
6	Make your driver stand alone. **Remove** (or comment out) the include statement in the Nt32pkg.fdf file: `#INF  MyWizardDriver/MyWizardDriver.inf` `INF  MdeModulePkg/Universal/Network/IScsiD` `#INF  MyWizardDriver/MyWizardDriver.inf` `!if $(BUILD_NEW_SHELL) == TRUE` `INF  ShellPkg/Application/Shell/Shell.inf` `!endif`
7	**Save** Nt32pkg.fdf
8	**Update** the MyWizardDriver.uni file
9	Add the following code to the top of the file at approx. line 14 as shown: `#langdef fr-FR "Francais"` `12` `13 #langdef en "English"` `14 #langdef fr-FR "Francais"` `15` `16 #string STR_SAMPLE_FORM_SET_TITLE #language en "My Wizard` `17 #string STR_SAMPLE_FORM_SET_HELP #language en "Help for S` `18 #string STR_SAMPLE_FORM1_TITLE #language en "My Wizard` `19`
10	Add the following code to the end of the file: `#string STR_LANGUAGE_TEST_STRING #language en "Laurie's Test String"` `#language fr-FR "Chaîne de test de Laurie"` `75` `76 #string STR_MY_TIME_PROMPT #language en "My System Time"` `77` `78 #string STR_LANGUAGE_TEST_STRING #language en "Laurie's Test String"` `79 #language fr-FR "Chaîne de test de Laurie"` `80`
11	**Save** MyWizardDriver.uni
12	**Update** the MyWizardDriver.c file
13	**Add** the following local variable for `StringPtr` after "`BOOLEAN ActionFlag;`" and before "`Status = EFI_SUCCESS;`"(as shown below): `EFI_STRING StringPtr;`

Step	Action
	
14	Add the following code after "`FreePool (ConfigRequestHdr);`" (as shown below) to edit the driver's entry point with a debug and print statement by making a call to the HiiGetString for the token to print (at approx line 364): `StringPtr = HiiGetString (HiiHandle[0], STRING_TOKEN (STR_LANGUAGE_TEST_STRING), NULL);` `DEBUG ((EFI_D_INFO,"[MyWizardDriver-Entrypoint] My String was: %s\n", StringPtr) );` `Print(L"%s\n", StringPtr );` 
15	**Save** the MyWizardDriver.c
16	In the Visual Studio Command Prompt, **type** `build`
17	**Press** "Enter"
18	**Type** `build run`
19	**Press** "Enter"
20	**Type** `Fs0:`
21	**Press** "Enter"
22	**Type** `load MyWizardDriver.efi` and notice that the string's English version is displayed: 
23	**Type** `Reset` 

Step	Action
24	**Press** "Enter"
25	**Type** `build`
26	**Type** `build run`
27	**Press** "Enter"
28	**Type** `exit` at the shell prompt
29	**Select Language** 
30	**Press** "Enter"
31	 **Select** "Français"
32	**Press** "Enter"
33	 **Select** "Continuer"
34	**Press** "Enter"
35	At the Shell Prompt, **type** `Fs0:`

Step	Action
36	**Type** `load MyWizardDriver.efi`  Notice that the string's French version is displayed:
37	**Type "reset"** 
38	**Press** "Enter" to return to the Visual Studio Command Prompt 

For any build issues copy the solution files from C:\Fw\LabSolutions\Lesson9.11
NOTE: Del Directory C:\fw\edk2\Build\NT32\DEBUG_VS2010x86\IA32\MyWizardDriver before the Build command to build the MyWizardDriver Clean.
Make sure you update Nt32Pkg.fdf.

LESSON 12
EDK II DEBUGGING

12.1 Adding Debug Statements

In this lab, you'll learn how to add debug statements. This lab uses code from a previous exercise as a starting point (refer to 7.2 Writing Simple UEFI Applications). Before proceeding, verify that the SampleApp code is present in your workspace and that the code references the Nt32Pkg.dsc file.

NOTE: Complete the following steps if you have not completed Lab 7.2:

Step		Action

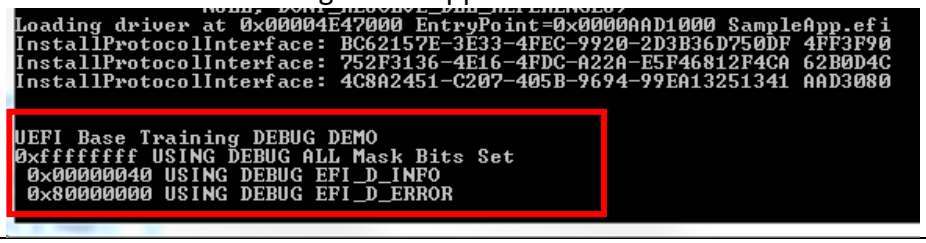
1	**Skip if Lab 7.2 Completed**	Create a Directory under the workspace `c:\fw\edk2` **`SampleApp`**
2		Now**, locate** and open the directory: `C:\fw\LabSampleCode\SampleAppDebug`
3		**Copy** the following files to `C:\fw\edk2\SampleApp` 1. SampleApp.c 2. SampleApp.inf
4		**Open and Edit** C:\Fw\edk2\Nt32Pkg\Nt32Pkg.dsc
5		**Add** the following line **above** the `[BuildOptions]` section of Nt32Pkg.dsc: `SampleApp/SampleApp.inf`

Step	Action
1	Open a Visual Studio Command Prompt and **type** `cd c:\fw\edk2`
2	**Press** "Enter"
3	**Type** `edksetup`
4	**Press** "Enter" to setup the EDK II environment.
5	**Open** C:\FW\edk2\SampleApp\SampleApp.c
6	**Add** the following to the include statements at the top of the file:
	`#include <Library/DebugLib.h>`
	`#include <Uefi.h>` `#include <Library/UefiApplicationEntryPoint.h>` `#include <Library/UefiLib.h>` `#include <Library/UefiBootServicesTableLib.h>` `#include <Library/BaseMemoryLib.h>` `#include <Library/DebugLib.h>` `#define CHAR_DOT 0x002E // '.' in Unicode`
7	**Locate** the `UefiMain` function**.**
8	**Add (copy and paste)** the following code after the "`EFI_INPUT_KEY KEY;`" statement: and before the first print statement as shown in the screen shot below:

Step	Action

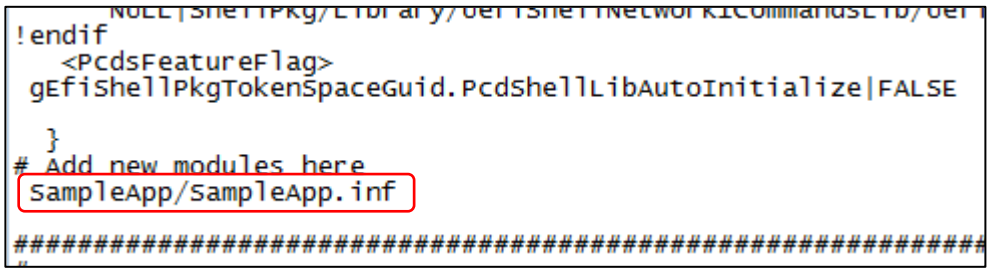
```
DEBUG ((0xffffffff, "\n\nUEFI Base Training DEBUG DEMO\n") );
DEBUG ((0xffffffff, "0xffffffff USING DEBUG ALL Mask Bits Set\n") );

DEBUG ((EFI_D_INIT,      " 0x%08x USING DEBUG EFI_D_INIT\n" ,
(UINTN)(EFI_D_INIT))  );
DEBUG ((EFI_D_WARN,      " 0x%08x USING DEBUG EFI_D_WARN\n",
(UINTN)(EFI_D_WARN))  );
DEBUG ((EFI_D_LOAD,      " 0x%08x USING DEBUG EFI_D_LOAD\n",
(UINTN)(EFI_D_LOAD))  );
DEBUG ((EFI_D_FS,        " 0x%08x USING DEBUG EFI_D_FS\n", (UINTN)(EFI_D_FS))
);
DEBUG ((EFI_D_POOL,      " 0x%08x USING DEBUG EFI_D_POOL\n",
(UINTN)(EFI_D_POOL))  );
DEBUG ((EFI_D_PAGE,      " 0x%08x USING DEBUG EFI_D_PAGE\n",
(UINTN)(EFI_D_PAGE))  );
DEBUG ((EFI_D_INFO,      " 0x%08x USING DEBUG EFI_D_INFO\n",
(UINTN)(EFI_D_INFO))  );
DEBUG ((EFI_D_DISPATCH, " 0x%08x USING DEBUG
EFI_D_DISPATCH\n",(UINTN)(EFI_D_DISPATCH)));
DEBUG ((EFI_D_VARIABLE, " 0x%08x USING DEBUG
EFI_D_VARIABLE\n",(UINTN)(EFI_D_VARIABLE)));
DEBUG ((EFI_D_BM,        " 0x%08x USING DEBUG EFI_D_BM\n", (UINTN)(EFI_D_BM))
);
DEBUG ((EFI_D_BLKIO,     " 0x%08x USING DEBUG EFI_D_BLKIO\n",
(UINTN)(EFI_D_BLKIO))  );
DEBUG ((EFI_D_NET,       " 0x%08x USING DEBUG EFI_D_NET\n", (UINTN)(EFI_D_NET))
);
DEBUG ((EFI_D_UNDI,      " 0x%08x USING DEBUG EFI_D_UNDI\n",
(UINTN)(EFI_D_UNDI))  );
DEBUG ((EFI_D_LOADFILE, " 0x%08x USING DEBUG
EFI_D_LOADFILE\n",(UINTN)(EFI_D_LOADFILE)));
DEBUG ((EFI_D_EVENT,     " 0x%08x USING DEBUG EFI_D_EVENT\n",
(UINTN)(EFI_D_EVENT))  );
DEBUG ((EFI_D_ERROR,     " 0x%08x USING DEBUG EFI_D_ERROR\n",
(UINTN)(EFI_D_ERROR))  );
```

```
EFI_STATUS
EFIAPI
UefiMain (
  IN EFI_HANDLE        ImageHandle,
  IN EFI_SYSTEM_TABLE  *SystemTable
  )
{
    UINTN          EventIndex;
    BOOLEAN        ExitLoop;
    EFI_INPUT_KEY      Key;

    DEBUG ((0xffffffff, "\n\nUEFI Base Training DEBUG DEMO\n") );
    DEBUG ((0xffffffff, "0xffffffff USING DEBUG ALL Mask Bits Set\r\n") );


    DEBUG ((EFI_D_INIT,      " 0x%08x USING DEBUG EFI_D_INIT\r\n" , (UINTN)(EFI_D_INIT))  );
    DEBUG ((EFI_D_WARN,      " 0x%08x USING DEBUG EFI_D_WARN\r\n", (UINTN)(EFI_D_WARN))  );
    DEBUG ((EFI_D_LOAD,      " 0x%08x USING DEBUG EFI_D_LOAD\r\n", (UINTN)(EFI_D_LOAD))  );
    DEBUG ((EFI_D_FS,        " 0x%08x USING DEBUG EFI_D_FS\r\n", (UINTN)(EFI_D_FS))  );
    DEBUG ((EFI_D_POOL,      " 0x%08x USING DEBUG EFI_D_POOL\r\n", (UINTN)(EFI_D_POOL))  );
    DEBUG ((EFI_D_PAGE,      " 0x%08x USING DEBUG EFI_D_PAGE\r\n", (UINTN)(EFI_D_PAGE))  );
    DEBUG ((EFI_D_INFO,      " 0x%08x USING DEBUG EFI_D_INFO\r\n", (UINTN)(EFI_D_INFO))  );
    DEBUG ((EFI_D_DISPATCH, " 0x%08x USING DEBUG EFI_D_DISPATCH\r\n", (UINTN)(EFI_D_DISPATCH))  );
    DEBUG ((EFI_D_VARIABLE, " 0x%08x USING DEBUG EFI_D_VARIABLE\r\n", (UINTN)(EFI_D_VARIABLE))  );
    DEBUG ((EFI_D_BM,        " 0x%08x USING DEBUG EFI_D_BM\r\n", (UINTN)(EFI_D_BM))  );
    DEBUG ((EFI_D_BLKIO,     " 0x%08x USING DEBUG EFI_D_BLKIO\r\n", (UINTN)(EFI_D_BLKIO))  );
    DEBUG ((EFI_D_NET,       " 0x%08x USING DEBUG EFI_D_NET\r\n", (UINTN)(EFI_D_NET))  );
    DEBUG ((EFI_D_UNDI,      " 0x%08x USING DEBUG EFI_D_UNDI\r\n", (UINTN)(EFI_D_UNDI))  );
    DEBUG ((EFI_D_LOADFILE, " 0x%08x USING DEBUG EFI_D_LOADFILE\r\n", (UINTN)(EFI_D_LOADFILE))  );
    DEBUG ((EFI_D_EVENT,     " 0x%08x USING DEBUG EFI_D_EVENT\r\n", (UINTN)(EFI_D_EVENT))  );
    DEBUG ((EFI_D_ERROR,     " 0x%08x USING DEBUG EFI_D_ERROR\r\n", (UINTN)(EFI_D_ERROR))  );
```

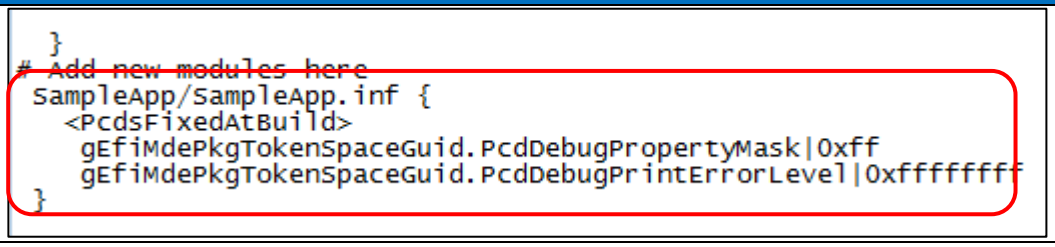
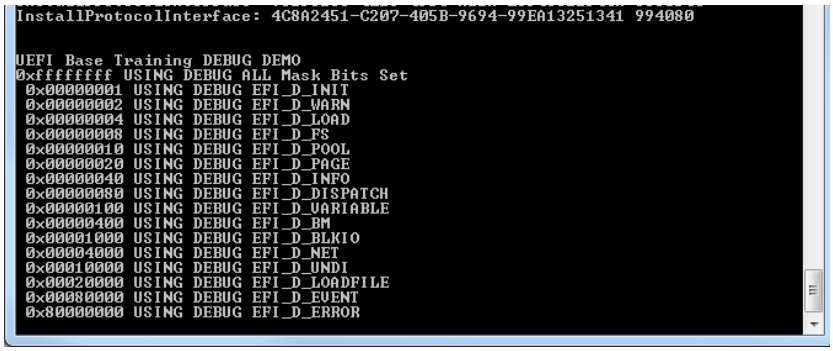
Step	Action
9	**Save and close the file.**
10	At the Visual Studio Command Prompt, **type** `build`
11	**Press** "Enter" to rebuild the NT32 project.
12	**Type** `build run`
13	**Press** "Enter" to start the NT32 Emulation
14	At the Shell 2.0 prompt, **type** `SampleApp`
15	**Press** "Enter" to run the new application. ***Note***: the DEBUG messages that appear in the Visual Studio Command Prompt: 
16	At the Shell 2.0 prompt, **type** `reset`
17	**Press** "Enter" to close the NT32 Emulation.

12.2 Changing DEBUG Features Using PCD Values

In this lab, you'll learn how to use PCD values to change debugging capabilities. The previous lab, 12.1 Adding Debug Statements, did not display all the DEBUG messages added to SampleApp.c. This lab shows how to change this behavior.

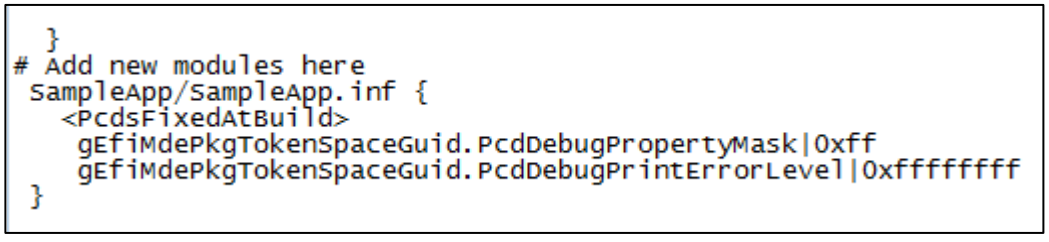
Step	Action
1	**Open** C:\fw\edk2\Nt32Pkg\Nt32Pkg.dsc
2	**Locate** the entry for `SampleApp/SampleApp.inf` 
3	**Replace** the "`SampleApp/SampleApp.inf`" entry with the following text:

```
SampleApp/SampleApp.inf {
   <PcdsFixedAtBuild>
    gEfiMdePkgTokenSpaceGuid.PcdDebugPropertyMask|0xff

gEfiMdePkgTokenSpaceGuid.PcdDebugPrintErrorLevel|0xffffffff
 }
```

Step	Action
	
4	**Save and close** the file.
5	In the Visual Studio Command Prompt, **type** **`build`**
6	**Press** "Enter" to rebuid the NT32 project.
7	**Type** `build run`
8	**Press** "Enter" to start the NT32 Emulation.
9	At the Shell 2.0 prompt, **type** SampleApp
10	**Press** "Enter" to run the program. ***Note***: Notice the changes in DEBUG output for SampleApp in the Visual Studio Command Prompt. Since the new PCD definitions were only applied to `SampleApp`, the DEBUG output properties are not changed for other parts of the NT32 project. 
11	At the Shell 2.0 prompt, **type** `reset`
12	**Press** "Enter" to close the NT32 Emulation.

12.3 Using Library Instances for Debugging

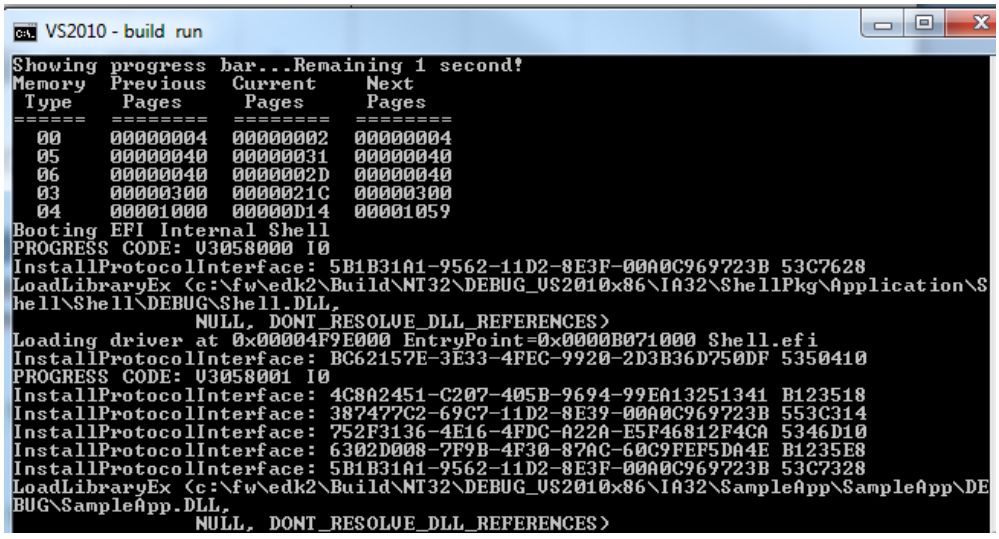
In this lab, you'll learn how to add specific debug library instances.

Step	Action
1	**Open** C:\fw\edk2\Nt32Pkg\Nt32Pkg.dsc
2	**Locate** the entry for **`SampleApp/SampleApp.inf`** 

Step	Action
3	**Replace** the "SampleApp/SampleApp.inf" entry with the following text: `SampleApp/SampleApp.inf { <LibraryClasses> DebugLib|MdePkg/Library/UefiDebugLibConOut/UefiDebugLibConOut.inf }` `# Add new modules here SampleApp/SampleApp.inf { <LibraryClasses> DebugLib|MdePkg/Library/UefiDebugLibConOut/UefiDebugLibConOut.inf }`
4	**Save and close** the file.
5	In the Visual Studio Command Prompt, **type** `build`
6	**Press** "Enter" to rebuid the NT32 project.
7	**Type** `build run`
8	**Press** "Enter" to start the NT32 Emulation.
9	At the Shell 2.0 prompt, **type** `SampleApp`
10	**Press** "Enter" to run the new version of the program. `Shell> sampleapp` `UEFI Base Training DEBUG DEMO` `0xffffffff USING DEBUG ALL Mask Bits Set` `0x00000040 USING DEBUG EFI_D_INFO` `0x80000000 USING DEBUG EFI_D_ERROR` `System Table: 0x0624BF90` `Press any Key to continue :` `Enter text. Include a dot ('.') in a sentence then <Enter> to exit:` `this is a test string with Debug testing.` `Shell> _` ***Note***: Some of the DEBUG output is redirected to the UEFI Shell instead of the Visual Studio Command Prompt. This debug library instance only applies to SampleApp and does not alter the general debug behavior of NT32.
11	At the Shell 2.0 prompt, **type** `reset`
12	**Press** "Enter" to close the NT32 Emulation.

12.4 Using NULL Library Instances for Debugging

In this lab, you will use a NULL library instance for debugging.

Step	Action
1	**Open** C:\fw\edk2\Nt32Pkg\Nt32Pkg.dsc
2	**Locate** the entry for `SampleApp/SampleApp.inf`
3	**Modify** the "`SampleApp/SampleApp.inf`" entry with the following text: `SampleApp/SampleApp.inf {` `  <LibraryClasses>` `DebugLib|MdePkg/Library/BaseDebugLibNull/BaseDebugLibNull.inf` `}`
4	**Save and close** the file.
5	In the Visual Studio Command Prompt, **type** `build`
6	**Press** "Enter" to rebuid the NT32 project.
7	**Type** `build run`
8	**Press** "Enter" to start the NT32 Emulation.
9	At the Shell 2.0 prompt, **type** `SampleApp`
10	Press "Enter" to run the new version of the program. ***Note***: The DEBUG output for `SampleApp` is no longer redirected to the UEFI Shell or the Visual Studio Command Prompt. This debug library instance only applies to `SampleApp` and does not alter the general debug behavior of NT32. 
11	At the Shell 2.0 prompt, **type** `reset`
12	**Press** "Enter" to close the NT32 Emulation.

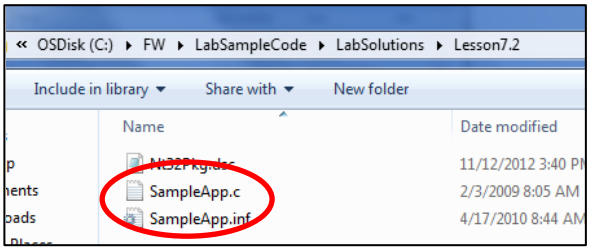
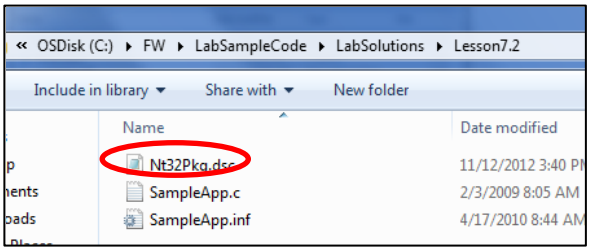
REFERENCE

Glossary of UEFI Terms and Acronyms

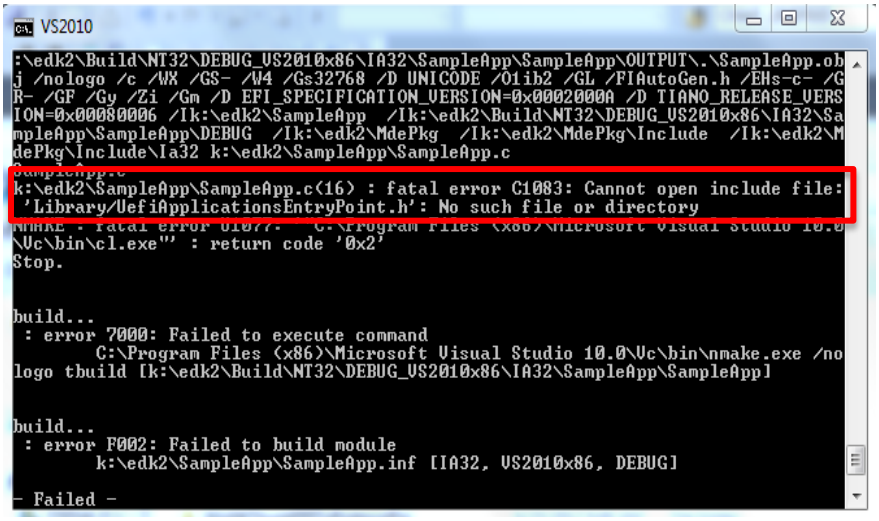
http://sourceforge.net/apps/mediawiki/tianocore/index.php?title=Acronyms

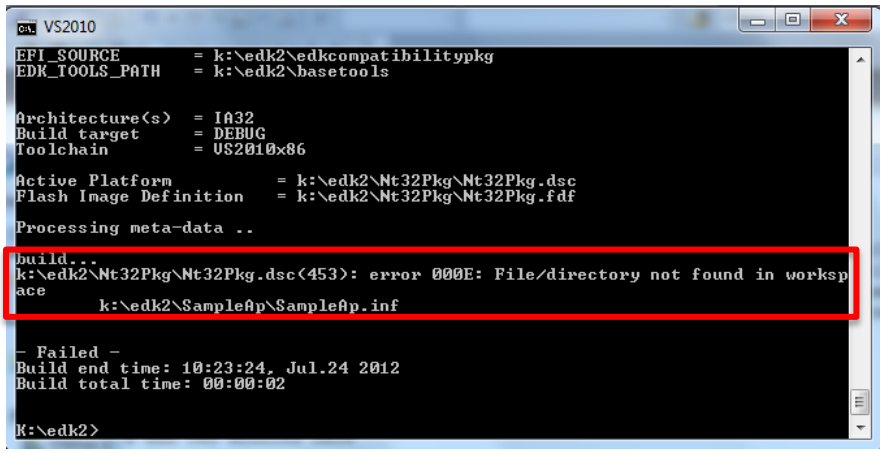
Lesson 7.2: Build Errors for Compiling NT32 without a Command Line Switch

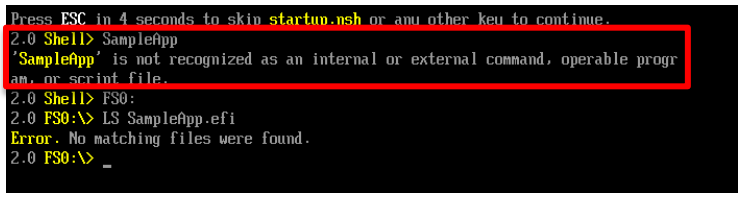
Lesson 7.2 Solution Files

Step	Action
1	**Locate** the solution files in C:\FW\LabSampleCode\LabSolutions\Lesson7.2
2	**Copy** SampleApp.c and SampleApp.inf to C:\fw\edk2\SampleApp 
3	Copy Nt32Pkg.dsc to C:\fw\edk2\Nt32Pkg 

Common Build Errors

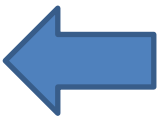






✓ Ensure .inf BaseName is SampleApp

✓ Check Nt32Pkg.dsc to ensure SampleApp.inf was included in components



Helpful Links

UEFI Forum	http://www.uefi.org
UEFI Open Source	http://www.tianocore.org
EFI Developer Kit (EDK II)	http://sourceforge.net/apps/mediawiki/tianocore/index.php?title=EDK2
EDK II Documents	http://sourceforge.net/projects/edk2/files/
UEFI Shell Documents	http://sourceforge.net/apps/mediawiki/tianocore/index.php?title=ShellPkg

Microsoft Windows and Visual Studio Matrix

Files **C:/fw/edk2/conf/target.txt** and **tools_def.txt** (Return to Configuring Build Tools)

Visual Studio	Version	WinXP	Win7 IA32	Win7 x64
VS2005	8.0	**target.txt** TOOL_CHAIN_TAG = VS2005 Requires WinDDK Link16**	**target.txt** TOOL_CHAIN_TAG = VS2005 Requires WinDDK Link16**	**target.txt** TOOL_CHAIN_TAG = **VS2005x86** Requires WinDDK Link16**
VS2008	9.0	**target.txt** TOOL_CHAIN_TAG = MYTOOLS Requires WinDDK Link16**	**target.txt** TOOL_CHAIN_TAG = MYTOOLS Requires WinDDK Link16**	**target.txt** TOOL_CHAIN_TAG = **VS2008x86**
VS2010	10.0	**target.txt** TOOL_CHAIN_TAG = **VS2010** **tools_def.txt** DEFINE WINSDK_VERSION = **v7.0A** Requires WinDDK Link16**	**target.txt** TOOL_CHAIN_TAG = **VS2010** **tools_def.txt** DEFINE WINSDK_VERSION = **v7.0A** Requires WinDDK Link16**	**target.txt** TOOL_CHAIN_TAG = **VS2010x86** **tools_def.txt** DEFINE WINSDK_VERSION = **v7.0A** DEFINE WINSDK_BIN = See A below
VS2012	11.0	Not supported	Not supported	**target.txt** TOOL_CHAIN_TAG = **VS2010x86** **tools_def.txt** DEFINE VS2010x86_BIN = See B below DEFINE VS2010x86_DLL = See C below DEFINE WINSDK_VERSION = **8.0** DEFINE WINSDK_BIN = See D below

A: C:\Program Files (x86)\Microsoft SDKs\Windows\DEF(WINSDK_VERSION)\bin

B: C:\Program Files (x86)\Microsoft Visual Studio 11.0\Vc\bin
C: C:\Program Files (x86)\Microsoft Visual Studio 11.0\Common7\IDE;DEF(VS2010x86_BIN)
D: C:\Program Files (x86)\Windows Kits\DEF(WINSDK_VERSION)\bin\x86
**WinDDK requires Link16.exe to be in C:\WinDDK\3790.1830\bin\bin16

Legal