



UEFI AND EDK II BASE TRAINING

*Lab and Reference Guide
Assumes Windows 10*

UEFI Driver Wizard – Adding HII	2
1. Adding Strings and Forms to Setup HII for User Configuration.....	3
a. Setup for Lab adding HII.....	3
b. Edit Driver for adding HII	6
2. Updating HII to Save Data Settings	20
3. Updating your driver to initialize data from the VFR data to the HII Database.....	33
a. Add HII Library Calls to Your Driver	33
b. Add your Driver to the platform	40
4. Updating the Menu: Reset Button	42
5. Updating the Menu: Pop-up Box.....	45
6. Updating the Menu: Creating a String to Name a Saved Configuration.....	51
7. Updating the Menu: Numeric Entry	58
8. Updating your Driver for Interactive Call Backs	63
a. Add the Case statements to the Call back routine.....	63
b. Update the Menu for Interactive items.....	66
9. Add code to your driver when Call Back events occur for Interactive Items	70
10. Adding an Additional Form Page	74
11. Adding Communication from Driver to Console through HII	82
Lab Setup	87
Pin Visual Studio Command Prompt for Windows	87
Preparing for the BUILD Command	87
Configuring Build Tools	89
Use the Microsoft Windows and Visual Studio Matrix	89
Microsoft Windows 10 – Visual Studio command prompt	90
Microsoft Windows and Visual Studio Matrix	94
Reference	95
Glossary of UEFI Terms and Acronyms	96
Helpful Links	96
Acknowledgement.....	97

UEFI Driver Wizard – Adding HII



1. Adding Strings and Forms to Setup HII for User Configuration

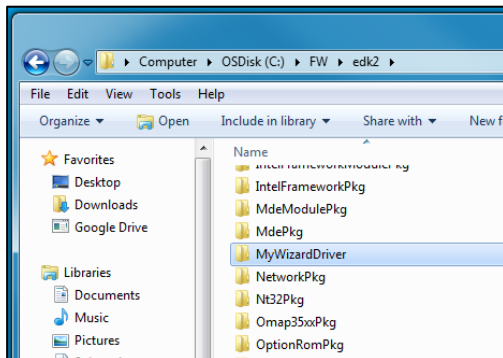
In this lab, you'll learn how to use HII to add strings and forms to a firmware setup menu for user configuration. Once you've complete this lab, your end result will match Figure 1.

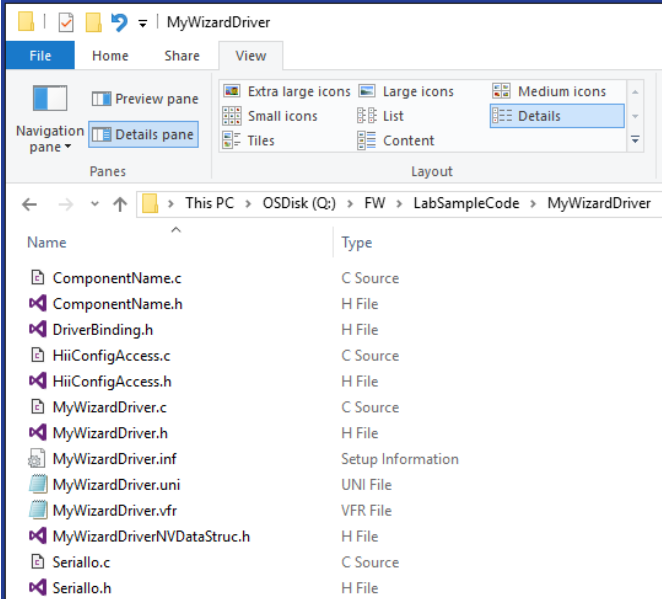
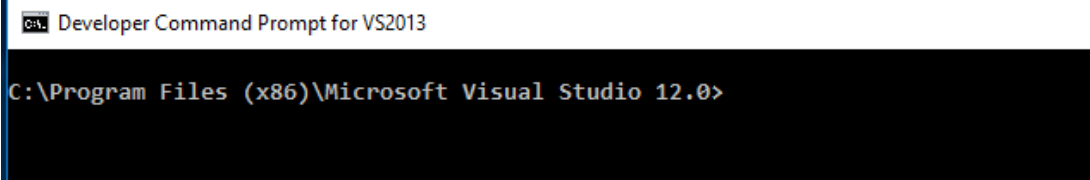
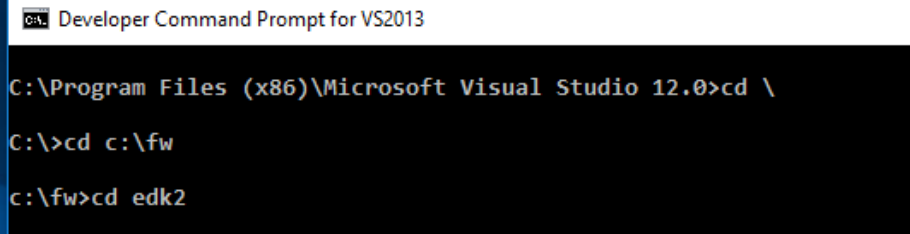


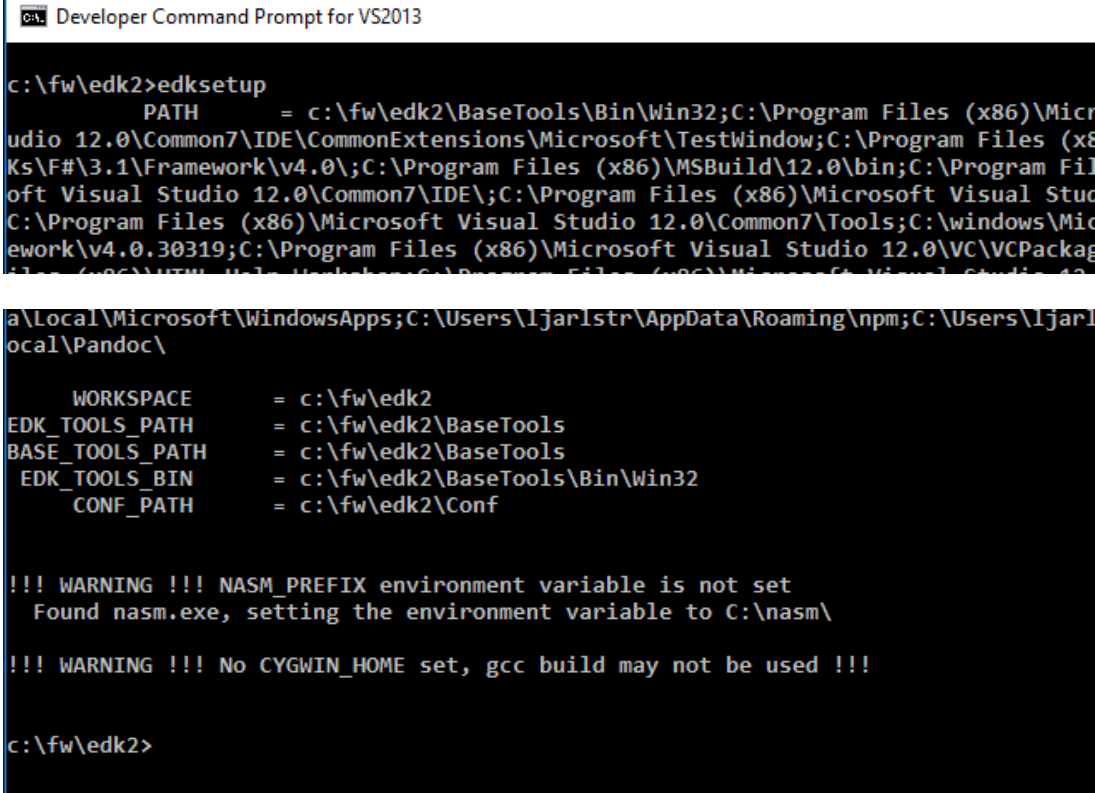
Figure 1 My Wizard Driver menu with strings and forms

a. Setup for Lab adding HII

Step	Action
1	Complete Lab Setup to configure for building with Visual Studio.
2	Start with LAB 6. on Driver porting Lab 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	<p>Copy the following Files to C:\FW\edk2\MyWizardDriver</p>  <p>The screenshot shows a Windows Explorer window titled 'MyWizardDriver'. The address bar indicates the path: 'This PC > OSDisk (C:) > FW > LabSampleCode > MyWizardDriver'. The file list shows the following files and their types:</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Type</th> </tr> </thead> <tbody> <tr> <td>ComponentName.c</td> <td>C Source</td> </tr> <tr> <td>ComponentName.h</td> <td>H File</td> </tr> <tr> <td>DriverBinding.h</td> <td>H File</td> </tr> <tr> <td>HiiConfigAccess.c</td> <td>C Source</td> </tr> <tr> <td>HiiConfigAccess.h</td> <td>H File</td> </tr> <tr> <td>MyWizardDriver.c</td> <td>C Source</td> </tr> <tr> <td>MyWizardDriver.h</td> <td>H File</td> </tr> <tr> <td>MyWizardDriver.inf</td> <td>Setup Information</td> </tr> <tr> <td>MyWizardDriver.uni</td> <td>UNI File</td> </tr> <tr> <td>MyWizardDriver.vfr</td> <td>VFR File</td> </tr> <tr> <td>MyWizardDriverNVDataStruc.h</td> <td>H File</td> </tr> <tr> <td>SerialIo.c</td> <td>C Source</td> </tr> <tr> <td>SerialIo.h</td> <td>H File</td> </tr> </tbody> </table>	Name	Type	ComponentName.c	C Source	ComponentName.h	H File	DriverBinding.h	H File	HiiConfigAccess.c	C Source	HiiConfigAccess.h	H File	MyWizardDriver.c	C Source	MyWizardDriver.h	H File	MyWizardDriver.inf	Setup Information	MyWizardDriver.uni	UNI File	MyWizardDriver.vfr	VFR File	MyWizardDriverNVDataStruc.h	H File	SerialIo.c	C Source	SerialIo.h	H File
Name	Type																												
ComponentName.c	C Source																												
ComponentName.h	H File																												
DriverBinding.h	H File																												
HiiConfigAccess.c	C Source																												
HiiConfigAccess.h	H File																												
MyWizardDriver.c	C Source																												
MyWizardDriver.h	H File																												
MyWizardDriver.inf	Setup Information																												
MyWizardDriver.uni	UNI File																												
MyWizardDriver.vfr	VFR File																												
MyWizardDriverNVDataStruc.h	H File																												
SerialIo.c	C Source																												
SerialIo.h	H File																												
5	<p>Open Visual Studio Command Prompt</p>  <p>The screenshot shows a 'Developer Command Prompt for VS2013' window. The command prompt displays the current directory: 'C:\Program Files (x86)\Microsoft Visual Studio 12.0>'. The title bar of the window reads 'C:\ Developer Command Prompt for VS2013'.</p>																												
6	<p>Type CD C:\fw\edk2</p>  <p>The screenshot shows the same 'Developer Command Prompt for VS2013' window. The command prompt displays the following sequence of commands to navigate to the target directory:</p> <pre>C:\Program Files (x86)\Microsoft Visual Studio 12.0>cd \ C:\>cd c:\fw c:\fw>cd edk2</pre>																												

Step	Action
7	<p>Type Edksetup (This is only needed if starting out with a new Visual Studio Command Prompt Window)</p>  <pre> c:\fw\edk2>edksetup PATH = c:\fw\edk2\BaseTools\Bin\Win32;C:\Program Files (x86)\Micro udio 12.0\Common7\IDE\CommonExtensions\Microsoft\TestWindow;C:\Program Files (x8 Ks\F#\3.1\Framework\v4.0\;C:\Program Files (x86)\MSBuild\12.0\bin;C:\Program Fil oft Visual Studio 12.0\Common7\IDE\;C:\Program Files (x86)\Microsoft Visual Stud C:\Program Files (x86)\Microsoft Visual Studio 12.0\Common7\Tools;C:\windows\Mic ework\v4.0.30319;C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\VCPackag iles (x86)\VFW_Help_Helpshar... a\Local\Microsoft\WindowsApps;C:\Users\ljarlstr\AppData\Roaming\npm;C:\Users\ljarl ocal\Pandoc\ WORKSPACE = c:\fw\edk2 EDK_TOOLS_PATH = c:\fw\edk2\BaseTools BASE_TOOLS_PATH = c:\fw\edk2\BaseTools EDK_TOOLS_BIN = c:\fw\edk2\BaseTools\Bin\Win32 CONF_PATH = c:\fw\edk2\Conf !!! WARNING !!! NASM_PREFIX environment variable is not set Found nasm.exe, setting the environment variable to C:\nasm\ !!! WARNING !!! No CYGWIN_HOME set, gcc build may not be used !!! c:\fw\edk2> </pre>

b. Edit Driver for adding HII

Step	Action
1	Open C:\fw\edk2\MyWizardDriver
2	Open the following files for updating: <ol style="list-style-type: none"> 1) MyWizardDriverNVDataStruc.h 2) MyWizardDriver.vfr 3) MyWizardDriver.uni 4) MyWizardDriver.h 5) MyWizardDriver.c 6) MyWizardDriver.inf
3	<p>Update the MyWizardDriverNVDataStruc.h file by copying and pasting the following GUID as shown below:</p> <p>This GUID is used to communicate to the HII Database and Browser Engine</p> <pre>#define MYWIZARDDRIVER_FORMSET_GUID \ { \ 0x5481db09, 0xe5f7, 0x4158, 0xa5, 0xc5, 0x2d, 0xbe, 0xa4, 0x95, 0x34, 0xff \ }</pre> <div> <pre> 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;</pre> </div>
4	Save MyWizardDriverNVDataStruc.h
5	<p>Update the MyWizardDriver.vfr file. Delete its contents and replace it with the following by copying and pasting:</p> <p>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.</p>

```
#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

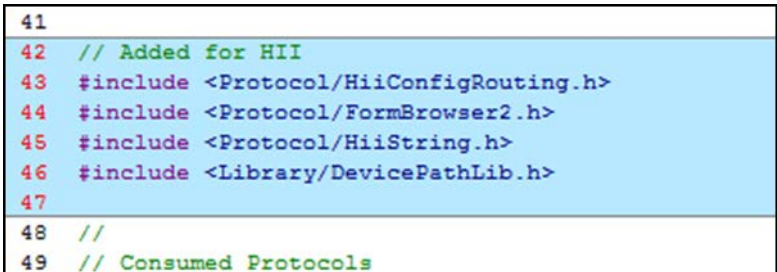
Save MyWizardDriver.uni

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()

```

```

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, ConfigAccess, MYWIZARDDRIVER_DEV_SIGNATURE)
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
115 ...

```

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
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

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`**

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 = MyWizardDriverHiiConfigAccessExtr
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 (
        mIfvVariableName,
        &mMyWizardDriverFormSetGuid,

```

```

        NULL,
        &BufferSize,
        &PrivateData->Configuration
    );
    if (EFI_ERROR (Status)) { // Not defined 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 defined 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

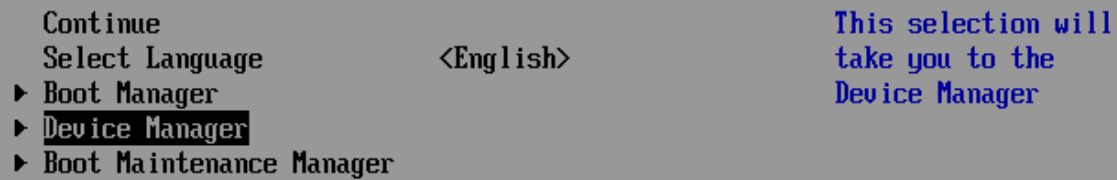
19

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	<div> <pre> 55 gEfiComponentNameProtocolGuid 56 gEfiHiiConfigAccessProtocolGuid 57 gEfiSimpleTextOutProtocolGuid 58 59 60 gEfiHiiStringProtocolGuid 61 gEfiHiiConfigRoutingProtocolGuid 62 gEfiFormBrowser2ProtocolGuid 63 gEfiHiiDatabaseProtocolGuid 64 </pre> </div>
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 2 building MyWizardDriver from the Driver Porting Lab)
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	<div> <pre> FS0:\> load MyWizardDriver.efi Image 'FS0:\MyWizardDriver.efi' loaded at 5EB9000 - Success FS0:\> exit_ </pre> </div> <p>Type exit</p>
32	Press "Enter"
33	Now at the setup front page menu, select "Device Manager"

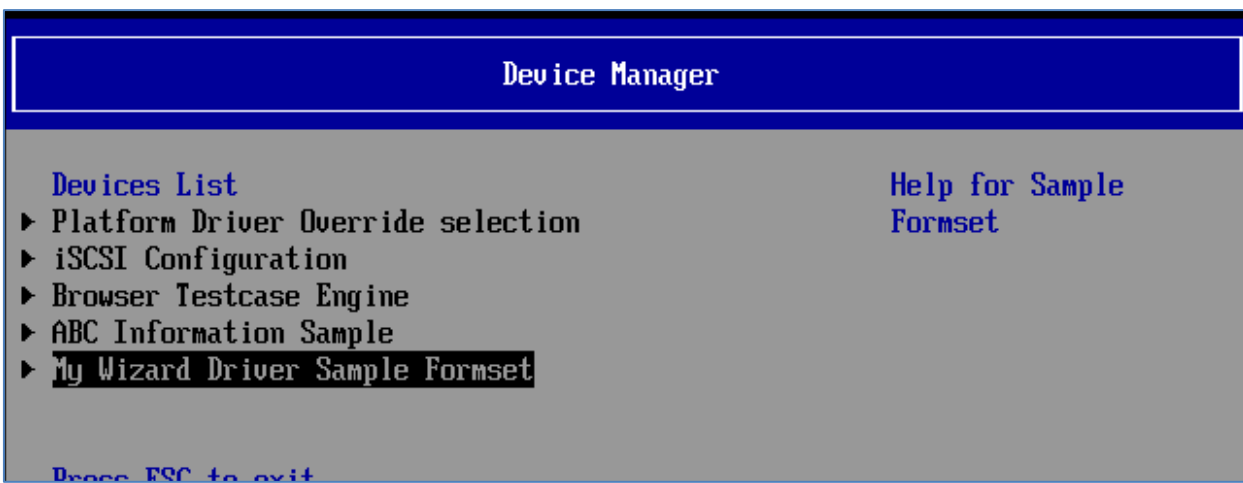


Continue
Select Language <English>
▶ Boot Manager
▶ **Device Manager**
▶ Boot Maintenance Manager

This selection will take you to the Device Manager

34 Press "Enter"

35 Inside the Device Manager menu **press** the down to "My Wizard Driver Sample Formset"



Device Manager

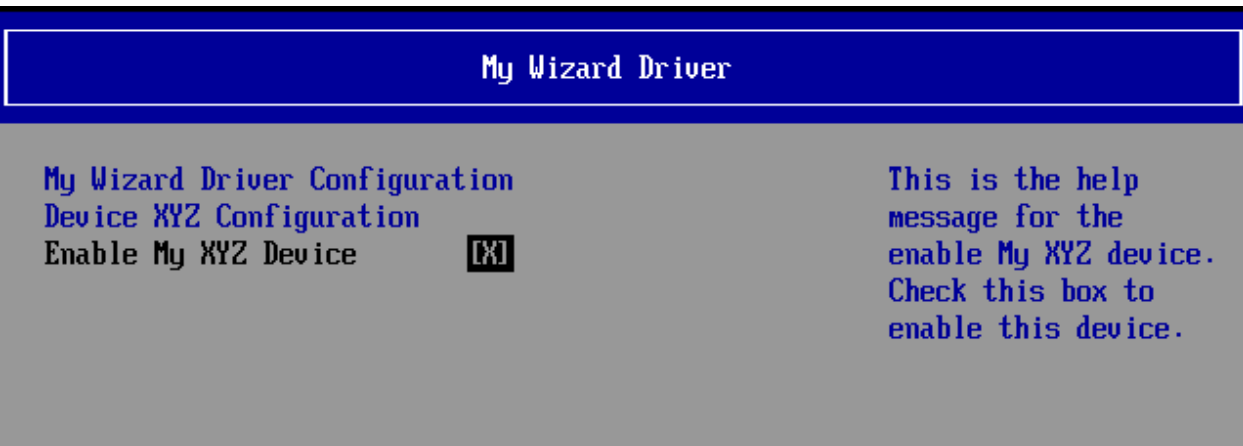
Devices List
▶ Platform Driver Override selection
▶ iSCSI Configuration
▶ Browser Testcase Engine
▶ ABC Information Sample
▶ **My Wizard Driver Sample Formset**

Help for Sample Formset

Press ESC to exit

Press "Enter" .

36



My Wizard Driver

My Wizard Driver Configuration
Device XYZ Configuration
Enable My XYZ Device ☒

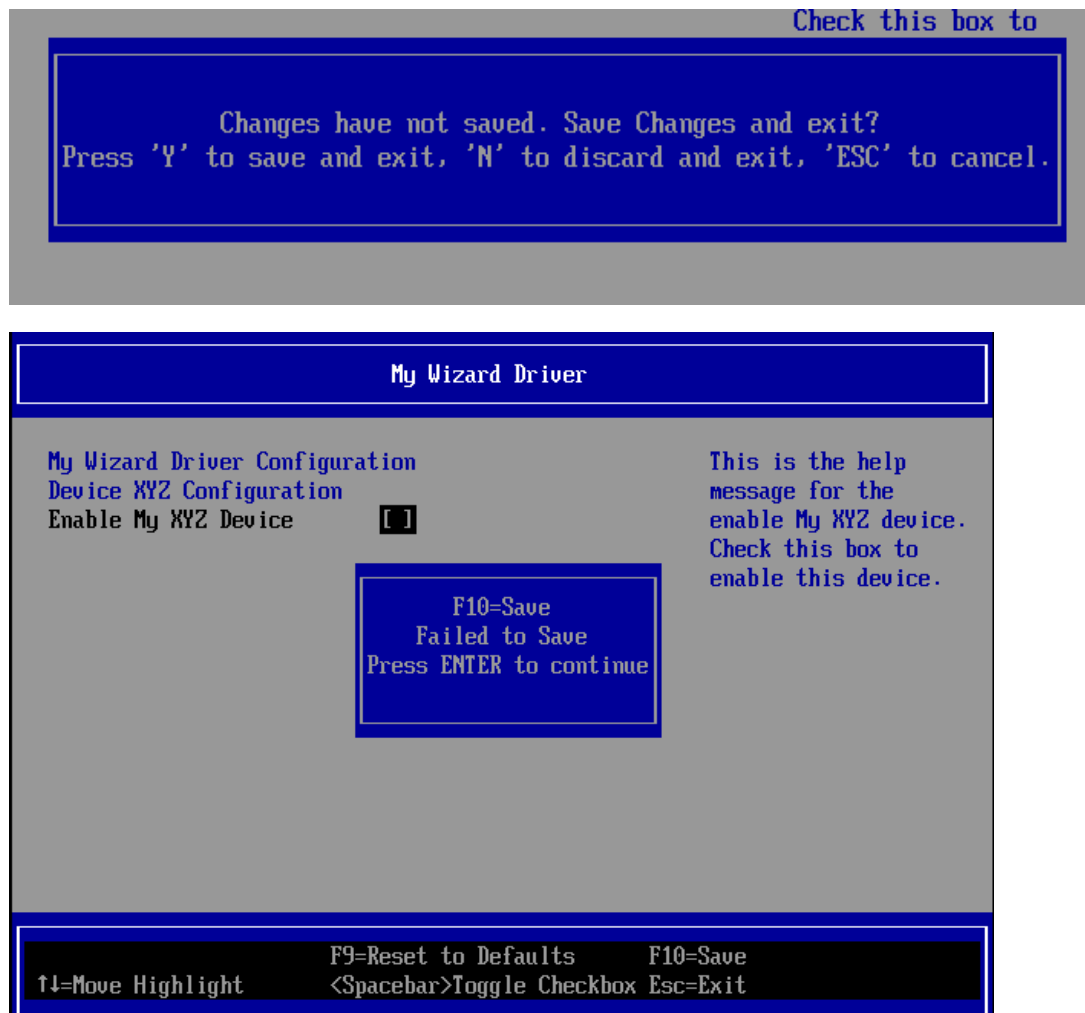
This is the help message for the enable My XYZ device. Check this box to enable this device.

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



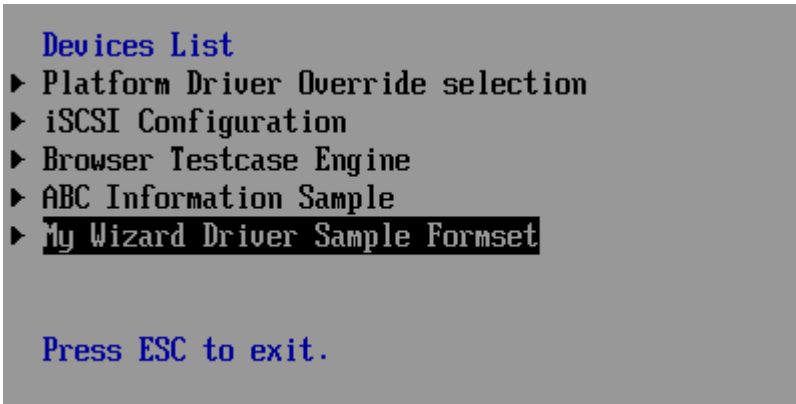
Note: You're not able to save the data changes at this point.

39 **Press** “Enter”

40 **Press** “Escape”, and then “Y” to exit

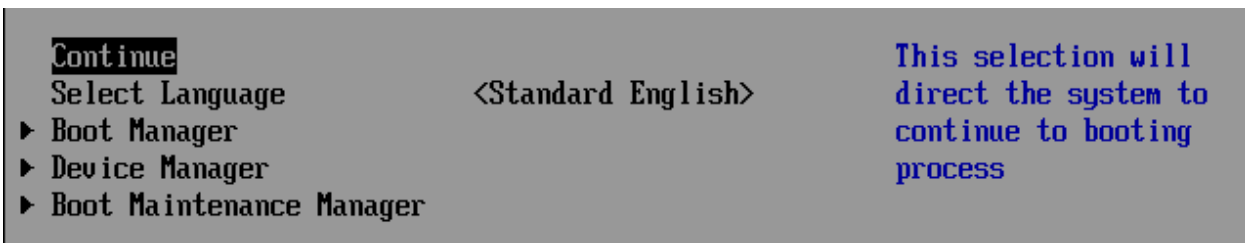
41

To Exit the “Device Manager” Page: Press “Escape”



42

Press Up Arrow to “Continue”

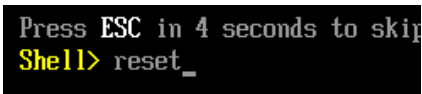


43

Press “Enter”

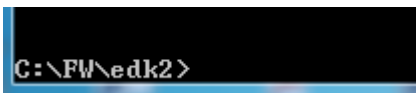
44

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\LessonE.1

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	<p>Update the MyWizardDriver.c file</p> <p>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.</p>
2	<p>Add the following local variable declarations in the function MyWizardDriverDriverEntryPoint Entry Point (as shown below Approx. line 185):</p> <pre> EFI_HII_STRING_PROTOCOL *HiiString; EFI_FORM_BROWSER2_PROTOCOL *FormBrowser2; EFI_HII_CONFIG_ROUTING_PROTOCOL *HiiConfigRouting; </pre> <div> <pre> 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 </pre> </div>
3	<p>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):</p> <p>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.</p>

Step	Action
	<pre> // // 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; </pre>

Step	Action
	<pre> 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, NULL, 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, NULL, 258 if (EFI_ERROR (Status)) { 259 return Status; 260 } 261 PrivateData->HiiConfigRouting = HiiConfigRouting; 262 263 // 264 // Publish sample Fromset and config access 265 // 266 // 267 Status = gBS->InstallMultipleProtocolInterfaces (</pre>

Step	Action
4	<p>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 <code>OpenProtocol</code> code with the <code>“//”</code> (approx. lines 289-298, as shown below) and add the comment <code>// Done above</code></p> <p>Make sure not to comment out the second <code>“ if (!EFI_ERROR (Status)) {”</code></p> <pre> 281 Status = gBS->OpenProtocol (282 ImageHandle, 283 &gEfiHiiPackageListProtocolGuid, 284 (VOID **)&PackageListHeader, 285 ImageHandle, 286 NULL, 287 EFI_OPEN_PROTOCOL_GET_PROTOCOL 288); 289 // Done above 290 // if (!EFI_ERROR (Status)) { 291 // // 292 // // Retrieve the pointer to the UEFI HII Database Protocol 293 // // 294 // Status = gBS->LocateProtocol (295 // &gEfiHiiDatabaseProtocolGuid, 296 // NULL, 297 // (VOID **)&HiiDatabase 298 //); 299 // if (!EFI_ERROR (Status)) { </pre> <p>Note: The earlier <code>LocateProtocol</code> code already found the pointer to the Hii Database protocol and stored it to the local pointer variable <code>HiiDatabase</code>.</p> <p>When we added the driver-consumed protocols, we searched via <code>LocateProtocol</code> for the Hii Database pointer function. Since we did it above we're now commenting out this code.</p>
5	<p>Comment out the matching “}” with “//” to the if statement (as shown below at approx. line 310):</p> <pre> 299 if (!EFI_ERROR (Status)) { 300 // 301 // Register list of HII packages in the HII Database 302 // 303 Status = HiiDatabase->NewPackageList (304 HiiDatabase, 305 PackageListHeader, 306 mDriverHandle[0], 307 &HiiHandle[0] 308); 309 ASSERT_EFI_ERROR (Status); 310 // } 311 } 312 Status = EFI_SUCCESS; 313 </pre>
6	<p>Save <code>MyWizardDriver.c</code></p>

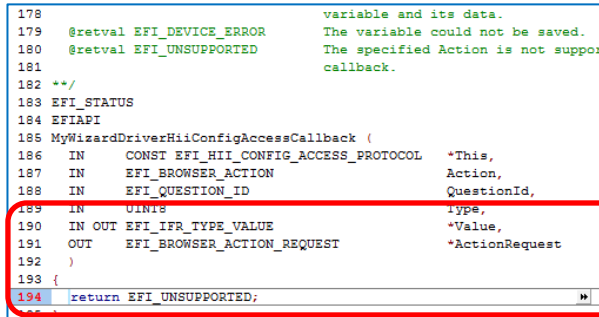
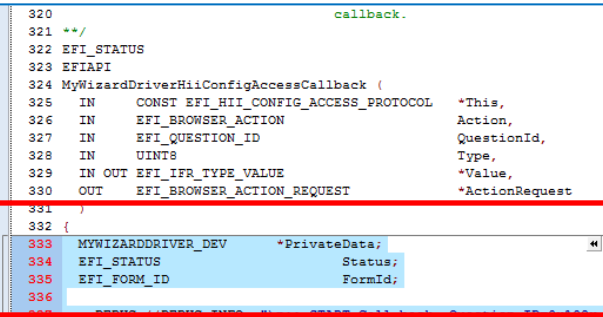
Step	Action
7	<p>Open C:\fw\edk2\MyWizardDriver\HiiConfigAccess.c.</p> <p>The Driver Wizard only made dummy functions for the extract, route and callback functions. In order to save the Data passed into the forms from the Hii Browser engine, you will need to port these functions to be functional.</p>
8	<p>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 :</p> <pre>extern EFI_GUID mMyWizardDriverFormSetGuid; extern CHAR16 mIfvVariableName[];</pre> <pre>12 #include "MyWizardDriver.h" 13 14 extern EFI_GUID mMyWizardDriverFormSetGuid; 15 extern CHAR16 mIfvVariableName[]; 16 17 18 /// 19 /// HII Config Access Protocol instance</pre>
9	<p>Locate MyWizardDriverHiiConfigAccessExtractConfig and replace line 108, “return EFI_NOT_FOUND”, with the following code spread over two pages:</p> <div> <p>FROM:</p> <pre>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 /**</pre> <p>TO:</p> <pre>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;</pre> </div>

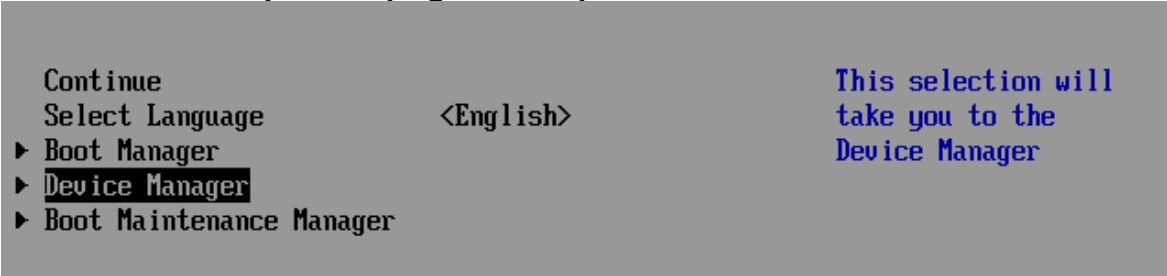
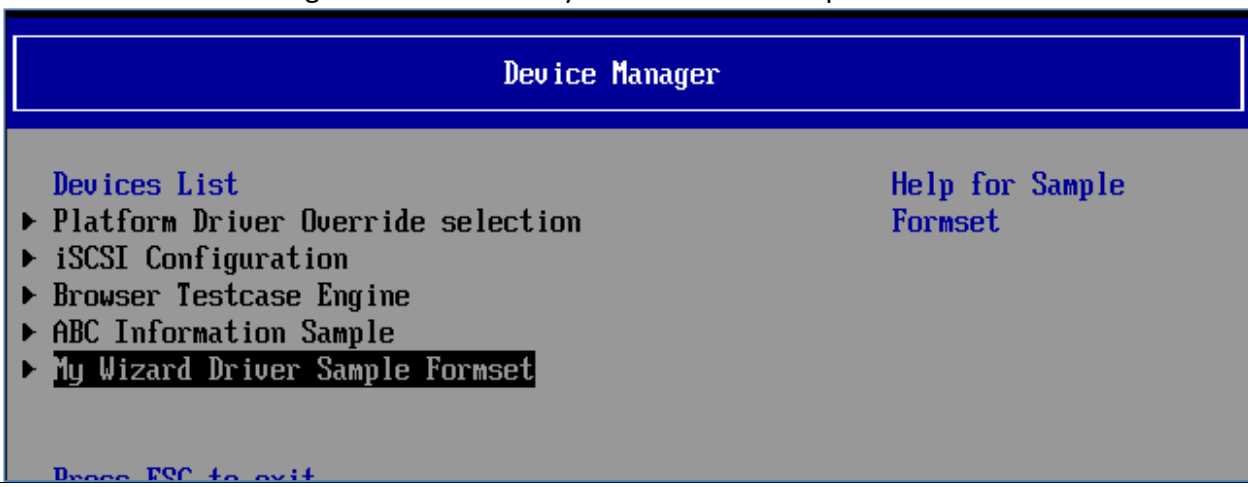
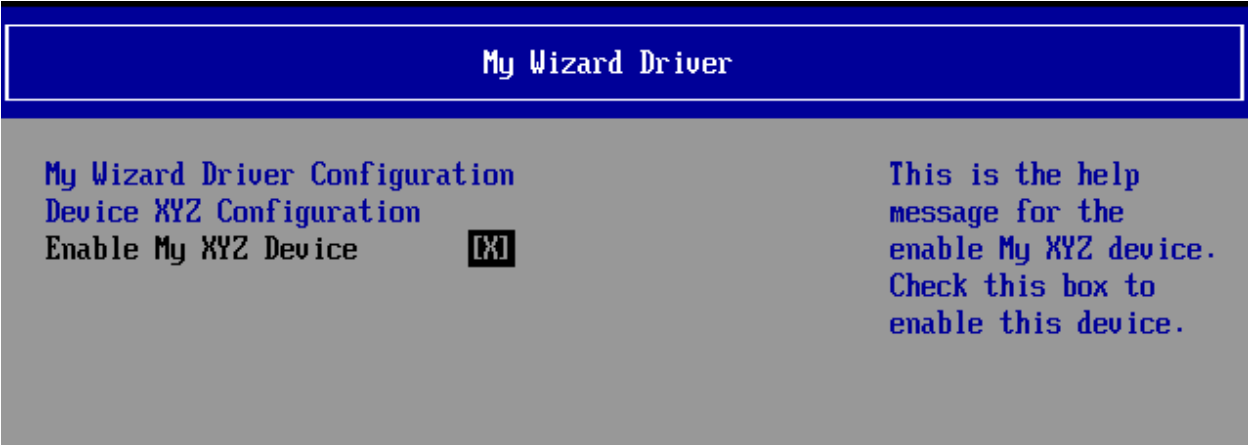
Step	Action
	<pre> EFI_STATUS Status; UINTN BufferSize; MYWIZARDDRIVER_DEV *PrivateData; EFI_HII_CONFIG_ROUTING_PROTOCOL *HiiConfigRouting; EFI_STRING ConfigRequest; EFI_STRING ConfigRequestHdr; UINTN Size; BOOLEAN AllocatedRequest; if (Progress == NULL Results == NULL) { return EFI_INVALID_PARAMETER; } // // 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); </pre>

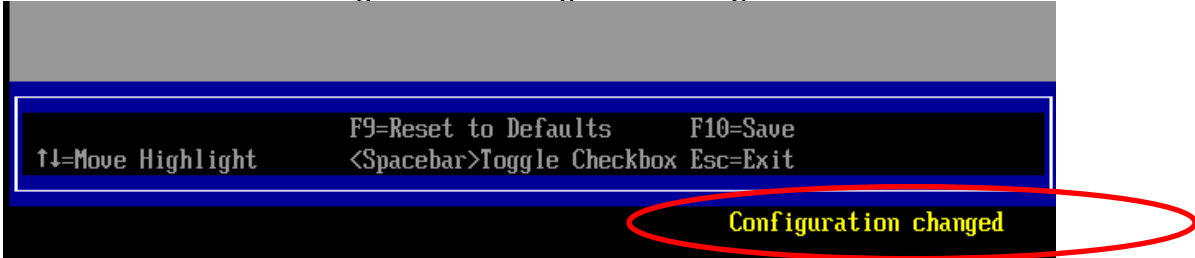
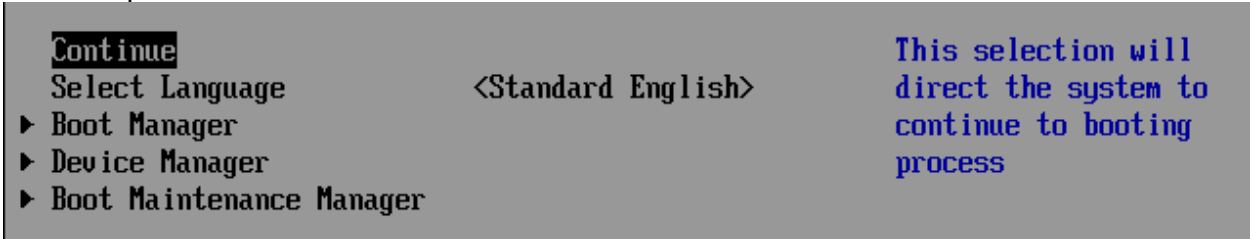
Step	Action
	<pre> if (EFI_ERROR (Status)) { return EFI_NOT_FOUND; } if (Request == NULL) { DEBUG ((DEBUG_INFO, "\n:: Inside of Extract Config and Request == Null ")); } else { ConfigRequest = Request; } // // Convert buffer data to <ConfigResp> by helper function BlockToConfig() // Status = HiiConfigRouting->BlockToConfig (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; </pre>
10	<p>Now locate <code>MyWizardDriverHiiConfigAccessRouteConfig</code> and replace line at approx. 228, <code>"return EFI_NOT_FOUND"</code>, with the following code:</p>

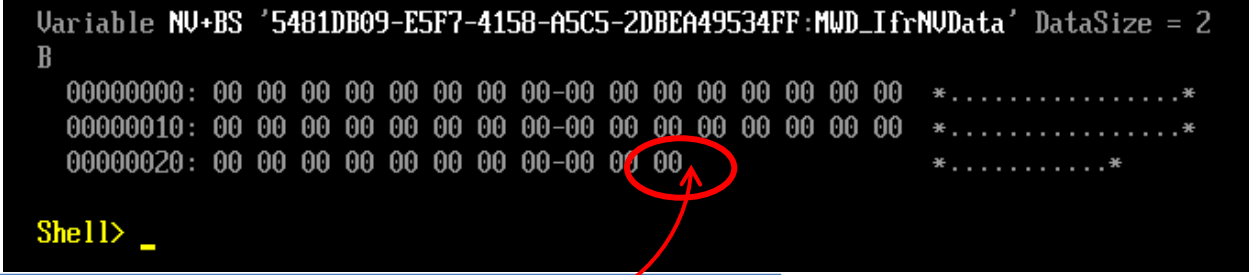
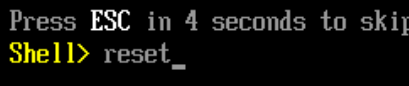

Step	Action
	<div> <div>FROM:</div> <div>TO:</div> </div> <div> <pre> 146 **/ 147 EFI_STATUS 148 EFIAPI 149 MyWizardDriverHiiConfigAccessRouteConfig (150 IN CONST EFI_HII_CONFIG_ACCESS_PROTOCOL *This, 151 IN CONST EFI_STRING Configuration, 152 OUT EFI_STRING *Progress 153) 154 { 155 return EFI_NOT_FOUND; 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 162 which data is either being passed back or being asked for </pre> </div> <div> <pre> 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 } </pre> </div>

Step	Action
	<pre> 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; </pre>

Step	Action
11	<p>Lastly, locate <code>MyWizardDriverHiiConfigAccessCallback</code> and replace at approx. line 326, "return <code>EFI_UNSUPPORTED</code>;", with the following code:</p> <p>FROM: </p> <p>TO: </p> <pre> 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; </pre>
12	Save <code>HiiConfigAccess.c</code>
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 <code>MyWizardDriver.efi</code>
20	Press "Enter"

Step	Action
21	<pre> FS0:\> load MyWizardDriver.efi Image 'FS0:\MyWizardDriver.efi' loaded at 5EB9000 - Success FS0:\> exit_ </pre> <p>Type <code>exit</code></p>
22	Press "Enter"
23	<p>Now at the setup front page menu press the down arrow to "Device Manager"</p> 
24	Press "Enter"
25	<p>Inside the Device Manager menu select "My Wizard Driver Sample Formset"</p> 
26	<p>Press "Enter" .</p> 

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	<p>Notice that enable is selected and saved in NVRam as the value of 0x00:</p>  <pre>Variable NV+BS '5481DB09-E5F7-4158-A5C5-2DBEA49534FF:MWD_IfrNVData' DataSize = 2 B 00000000: 00 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 00 *.....* 00000020: 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 *.....*</pre> <p>Shell> _</p> <pre>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;</pre> <p>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.</p> <p>Notice that the enable/disable byte is the last byte in data structure MWD_IfrNVData.MyWizardDriverChooseToEnable where 00 == disable and 01 == enable.</p>
37	<p>Type Reset</p>  <pre>Press ESC in 4 seconds to skip Shell> reset_</pre>
38	<p>Press "Enter" to return to the Visual Studio Command Prompt</p>  <pre>C:\FW\edk2></pre>

For any build issues copy the solution files from C:\Fw\LabSolutions\LessonE.2

NOTE: Del Directory C:\fw\edk2\Build\NT32IA32\DEBUG_VS2010x86\IA32\MyWizardDriver before the Build command to build the MyWizardDriver Clean

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](#)".

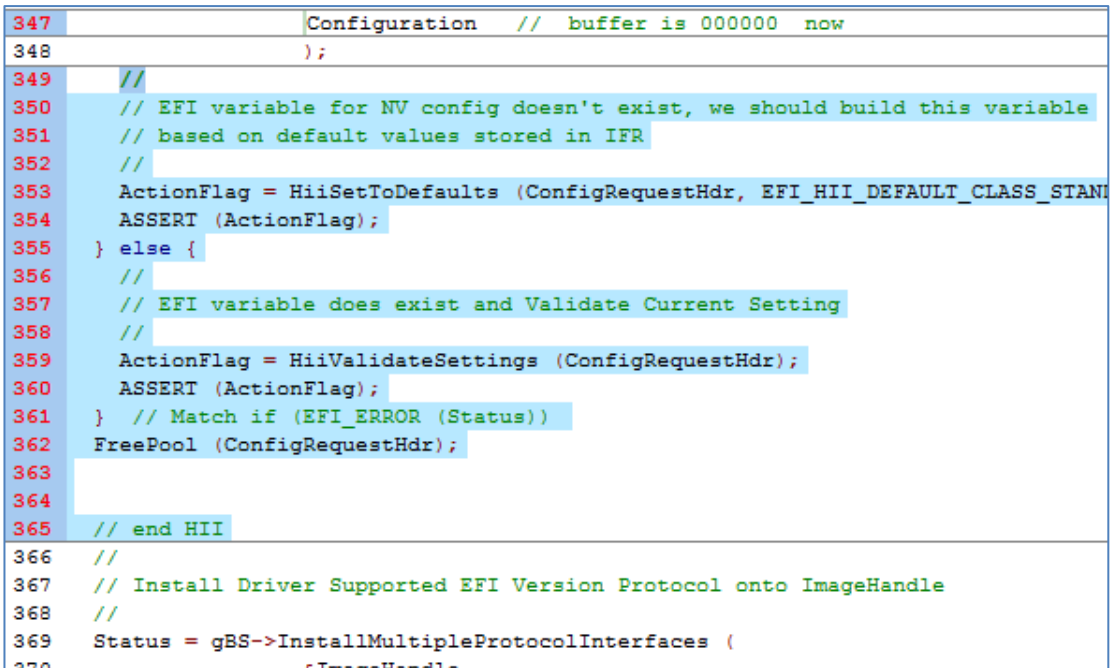
a. Add HII Library Calls to Your Driver

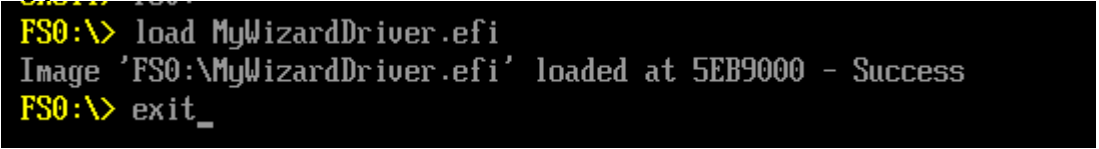
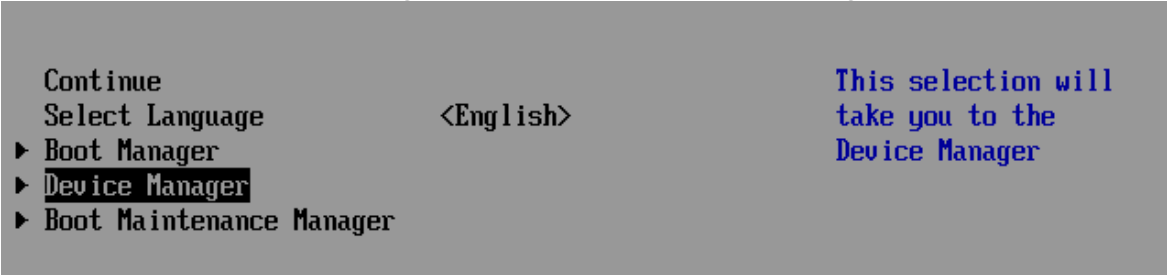
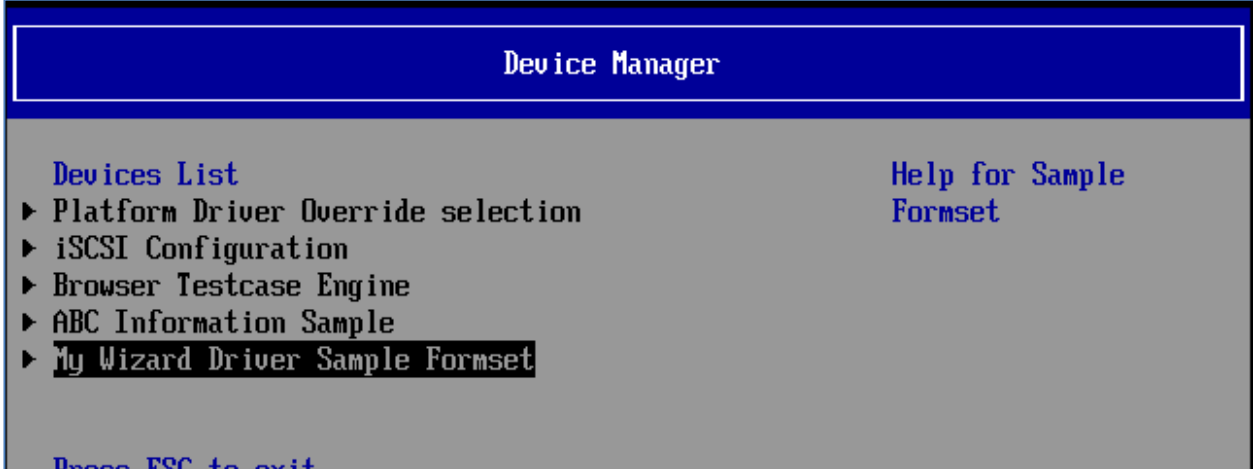
For this lab you will update the following files: MyWizardDriver.inf, MyWizardDriver.h, and MyWizardDriver.c

Step	Action
1	Update the MyWizardDriver.inf file
2	<p>Add the following package (as shown below):</p> <p>The HII Library in the MdeModulePkg has many functions to help with Communication to/from the Hii Database and Hii forms. One function call <code>HiiSetToDefaults</code> will compare the default settings from the .VFR file and update the driver's configuration buffer according to the settings in the .VFR file.</p> <p>MdeModulePkg/MdeModulePkg.dec</p> <pre> 22 [Packages] 23 MdePkg/MdePkg.dec 24 MdeModulePkg/MdeModulePkg.dec </pre> <p>Note: For other functions from the HII Library, open the .chm file "MdeModulePkg Document.chm" and search for HiiLib.h.</p>
3	<p>Add the following library class (as shown below):</p> <p>HiiLib</p> <pre> 39 [LibraryClasses] 40 UefiDriverEntryPoint 41 UefiBootServicesTableLib 42 MemoryAllocationLib 43 BaseMemoryLib 44 BaseLib 45 UefiLib 46 DevicePathLib 47 DebugLib 48 HiiLib </pre>


Step	Action
4	Save MyWizardDriver.inf
5	Update the MyWizardDriver.h file
6	<p>Add the following code (as shown below):</p> <pre>#include <Library/HiiLib.h></pre> <div> <pre>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></pre> </div>
7	Save MyWizardDriver.h
8	Update the MyWizardDriver.c file
9	<p>Add Locals: first add 2 locals for your drivers configuration buffer and a boolean flag from the Hii Library calls</p> <p>Add the following at Approx. Line 190.</p> <pre>MYWIZARDDRIVER_CONFIGURATION *Configuration; BOOLEAN ActionFlag;</pre> <div> <pre>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</pre> </div>
10	<p>Add the following to the MyWizardDriverDriverEntryPoint entry point function to line 319, approximately after "BufferSize =" as shown below</p>

Step	Action				
	<pre> // // Initialize configuration data // Configuration = &PrivateData->Configuration; ZeroMem (Configuration, sizeof (MYWIZARDDRIVER_CONFIGURATION)); // // Try to read NV config EFI variable first // ConfigRequestHdr = HiiConstructConfigHdr (&mMyWizardDriverFormSetGuid, mIfcVariableName, mDriverHandle[0]); ASSERT (ConfigRequestHdr != NULL); </pre> <div data-bbox="224 789 1252 1251"> <pre> 317 318 BufferSize = sizeof (MYWIZARDDRIVER_CONFIGURATION); 319 // 320 // Initialize configuration data 321 // 322 Configuration = &PrivateData->Configuration; 323 ZeroMem (Configuration, sizeof (MYWIZARDDRIVER_CONFIGURATION)); 324 325 // 326 // Try to read NV config EFI variable first 327 // 328 ConfigRequestHdr = HiiConstructConfigHdr (&mMyWizardDriverFormSetGuid, mIfcVar 329 ASSERT (ConfigRequestHdr != NULL); 330 331 332 // IF driver is not part of the Platform then need to get/set defaults for the 333 Status = gRT->GetVariable (</pre> </div>				
11	<p>Modify the following lines:</p> <p>@~338: remove: "&PrivateData->" from the "&PrivateData->Configuration"</p> <p>@~342: remove line: ZeroMem (&PrivateData->Configuration, sizeof (MYWIZARDDRIVER_CONFIGURATION));</p> <p>@~347: remove: "&PrivateData->" from the "&PrivateData->Configuration"</p> <div data-bbox="224 1476 1562 1856"> <table border="0"> <thead> <tr> <th data-bbox="224 1476 885 1514">FROM</th> <th data-bbox="885 1476 1562 1514">TO</th> </tr> </thead> <tbody> <tr> <td data-bbox="224 1514 885 1856"> <pre> 317 318 // IF driver is not part of the Platform then need to get/set defaults for the N 319 Status = gRT->GetVariable (320 mIfcVariableName, 321 &mMyWizardDriverFormSetGuid, 322 NULL, 323 &BufferSize, 324 &PrivateData->Configuration 325); 326 if (EFI_ERROR (Status)) { // Not defined yet so add it to the NV Variables. 327 // zero out buffer 328 ZeroMem (&PrivateData->Configuration, sizeof (MYWIZARDDRIVER_CONFIGURAT 329 Status = gRT->SetVariable(330 mIfcVariableName, 331 &mMyWizardDriverFormSetGuid, 332 EFI_VARIABLE_NON_VOLATILE EFI_VARIABLE_BOOTSERVICE_ACCESS, 333 sizeof (MYWIZARDDRIVER_CONFIGURATION), 334 &PrivateData->Configuration // buffer is 000000 now 335); 336 } 337 // </pre> </td> <td data-bbox="885 1514 1562 1856"> <pre> 330 331 332 // IF driver is not part of the Platform then need to get/set defaults for t 333 Status = gRT->GetVariable (334 mIfcVariableName, 335 &mMyWizardDriverFormSetGuid, 336 NULL, 337 &BufferSize, 338 Configuration 339); 340 if (EFI_ERROR (Status)) { // Not defined yet so add it to the NV Variables. 341 // zero out buffer 342 Status = gRT->SetVariable(343 mIfcVariableName, 344 &mMyWizardDriverFormSetGuid, 345 EFI_VARIABLE_NON_VOLATILE EFI_VARIABLE_BOOTSERVICE_ACCESS, 346 sizeof (MYWIZARDDRIVER_CONFIGURATION), 347 Configuration // buffer is 000000 now 348); 349 // 350 // EFI variable for NV config doesn't exist, we should build this variable </pre> </td> </tr> </tbody> </table> </div>	FROM	TO	<pre> 317 318 // IF driver is not part of the Platform then need to get/set defaults for the N 319 Status = gRT->GetVariable (320 mIfcVariableName, 321 &mMyWizardDriverFormSetGuid, 322 NULL, 323 &BufferSize, 324 &PrivateData->Configuration 325); 326 if (EFI_ERROR (Status)) { // Not defined yet so add it to the NV Variables. 327 // zero out buffer 328 ZeroMem (&PrivateData->Configuration, sizeof (MYWIZARDDRIVER_CONFIGURAT 329 Status = gRT->SetVariable(330 mIfcVariableName, 331 &mMyWizardDriverFormSetGuid, 332 EFI_VARIABLE_NON_VOLATILE EFI_VARIABLE_BOOTSERVICE_ACCESS, 333 sizeof (MYWIZARDDRIVER_CONFIGURATION), 334 &PrivateData->Configuration // buffer is 000000 now 335); 336 } 337 // </pre>	<pre> 330 331 332 // IF driver is not part of the Platform then need to get/set defaults for t 333 Status = gRT->GetVariable (334 mIfcVariableName, 335 &mMyWizardDriverFormSetGuid, 336 NULL, 337 &BufferSize, 338 Configuration 339); 340 if (EFI_ERROR (Status)) { // Not defined yet so add it to the NV Variables. 341 // zero out buffer 342 Status = gRT->SetVariable(343 mIfcVariableName, 344 &mMyWizardDriverFormSetGuid, 345 EFI_VARIABLE_NON_VOLATILE EFI_VARIABLE_BOOTSERVICE_ACCESS, 346 sizeof (MYWIZARDDRIVER_CONFIGURATION), 347 Configuration // buffer is 000000 now 348); 349 // 350 // EFI variable for NV config doesn't exist, we should build this variable </pre>
FROM	TO				
<pre> 317 318 // IF driver is not part of the Platform then need to get/set defaults for the N 319 Status = gRT->GetVariable (320 mIfcVariableName, 321 &mMyWizardDriverFormSetGuid, 322 NULL, 323 &BufferSize, 324 &PrivateData->Configuration 325); 326 if (EFI_ERROR (Status)) { // Not defined yet so add it to the NV Variables. 327 // zero out buffer 328 ZeroMem (&PrivateData->Configuration, sizeof (MYWIZARDDRIVER_CONFIGURAT 329 Status = gRT->SetVariable(330 mIfcVariableName, 331 &mMyWizardDriverFormSetGuid, 332 EFI_VARIABLE_NON_VOLATILE EFI_VARIABLE_BOOTSERVICE_ACCESS, 333 sizeof (MYWIZARDDRIVER_CONFIGURATION), 334 &PrivateData->Configuration // buffer is 000000 now 335); 336 } 337 // </pre>	<pre> 330 331 332 // IF driver is not part of the Platform then need to get/set defaults for t 333 Status = gRT->GetVariable (334 mIfcVariableName, 335 &mMyWizardDriverFormSetGuid, 336 NULL, 337 &BufferSize, 338 Configuration 339); 340 if (EFI_ERROR (Status)) { // Not defined yet so add it to the NV Variables. 341 // zero out buffer 342 Status = gRT->SetVariable(343 mIfcVariableName, 344 &mMyWizardDriverFormSetGuid, 345 EFI_VARIABLE_NON_VOLATILE EFI_VARIABLE_BOOTSERVICE_ACCESS, 346 sizeof (MYWIZARDDRIVER_CONFIGURATION), 347 Configuration // buffer is 000000 now 348); 349 // 350 // EFI variable for NV config doesn't exist, we should build this variable </pre>				

Step	Action
12	<p>Add the following code to the MyWizardDriverDriverEntryPoint entry point code at approximately line 349 before</p> <pre data-bbox="256 327 1446 426"> // // Install Driver Supported EFI Version Protocol onto ImageHandle // </pre> <p>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:</p> <pre data-bbox="219 590 1484 1203"> // // 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 </pre>  <pre data-bbox="219 1234 1323 1900"> 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_STANDARD); 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 (370 <ImageHandle </pre>

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  <pre> FS0:\> load MyWizardDriver.efi Image 'FS0:\MyWizardDriver.efi' loaded at 5EB9000 - Success FS0:\> exit_ </pre>
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	<p>Press “Enter” .</p> 
27	Press “Escape” to exit
28	<p>To Exit the “Device Manager” Page: Press “Escape”</p> 
29	<p>Press Up Arrow to “Continue”</p> 
30	Press “Enter”
31	<p>Type Reset</p> 

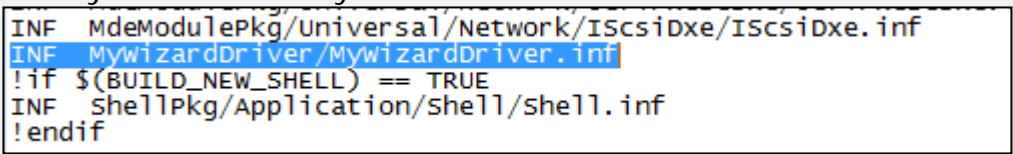
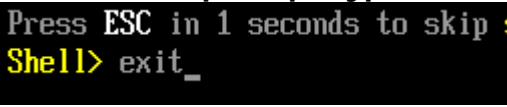
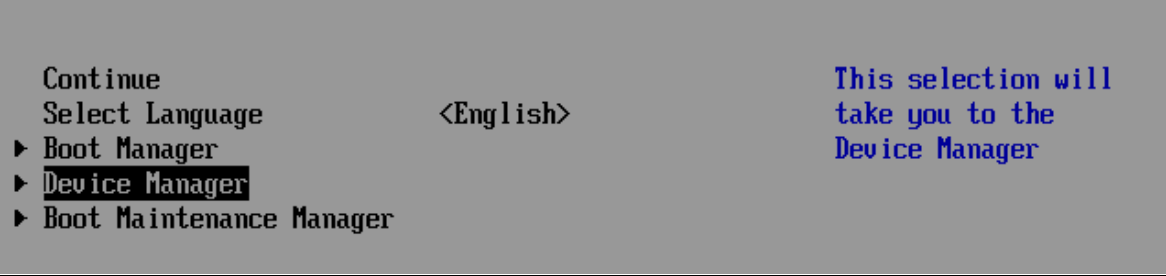
Step	Action
32	<p>Press “Enter” to return to the Visual Studio Command Prompt</p> 

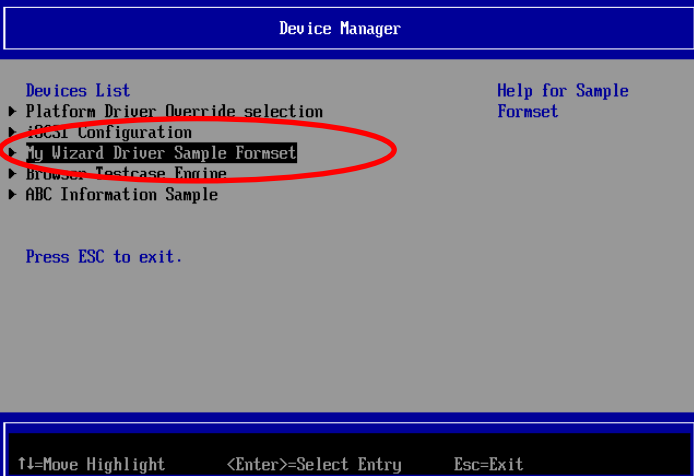
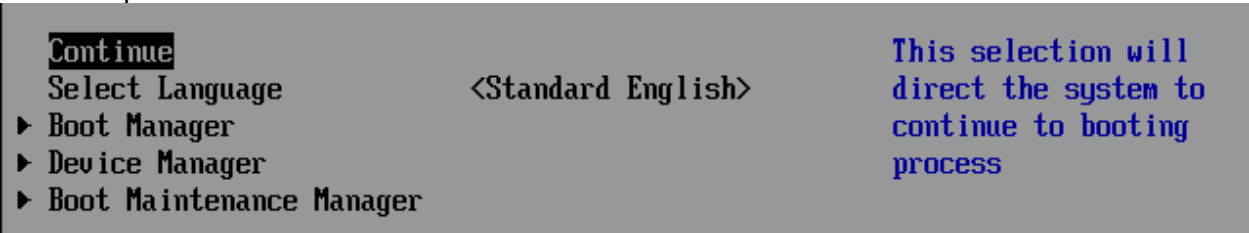
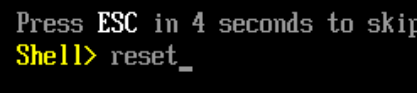

For any build issues copy the solution files from C:\Fw\LabSolutions\LessonE.3

NOTE: Del Directory C:\fw\edk2\Build\NT32IA32\DEBUG_VS2010x86\IA32\MyWizardDriver
before the Build command to build the MyWizardDriver Clean

b. 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 Add the following code (as shown below before " <code>!if \$(BUILD_NEW_SHELL) == TRUE</code> "):
2	INF MyWizardDriver/MyWizardDriver.inf  <pre> INF MdeModulePkg/Universal/Network/IScsiDxe/IScsiDxe.inf INF MyWizardDriver/MyWizardDriver.inf !if \$(BUILD_NEW_SHELL) == TRUE INF ShellPkg/Application/Shell/Shell.inf !endif </pre>
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  <pre> Press ESC in 1 seconds to skip Shell> exit_ </pre>
9	Press "Enter"
10	Now at the setup front page menu press the down arrow to "Device Manager"  <pre> Continue Select Language <English> └─ Boot Manager └─ Device Manager └─ Boot Maintenance Manager </pre> <p>This selection will take you to the Device Manager</p>
11	Press "Enter"

11	<p>Notice that the My Wizard Driver Sample Formset is added without having to issue the “Load” command from the shell prompt.</p>  <p>The screenshot shows the 'Device Manager' window with a 'Devices List' on the left. The list includes: Platform Driver Override selection, iSCSI Configuration, My Wizard Driver Sample Formset (highlighted with a red circle), Browser Testcase Engine, and ABC Information Sample. A 'Help for Sample Formset' link is on the right. At the bottom, it says 'Press ESC to exit.' and a legend: ↑=Move Highlight, <Enter>=Select Entry, Esc=Exit.</p>
12	<p>Press “Escape”</p>
13	<p>Press Up arrow to “Continue”</p>  <p>The screenshot shows a boot menu with 'Continue' highlighted. Other options are 'Select Language', 'Boot Manager', 'Device Manager', and 'Boot Maintenance Manager'. The language is set to '<Standard English>'. A note on the right states: 'This selection will direct the system to continue to booting process'.</p>
14	<p>Press “Enter”</p>
15	<p>Type Reset</p>  <p>The screenshot shows the UEFI Shell prompt 'Shell>' with the command 'reset_' being entered. A message above says 'Press ESC in 4 seconds to skip'.</p>
16	<p>Press “Enter” to return to the Visual Studio Command Prompt</p>  <p>The screenshot shows the Visual Studio Command Prompt with the path 'C:\FW\edk2' displayed.</p>



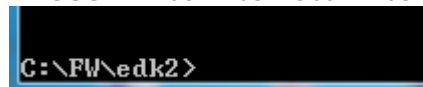
4. Updating the Menu: Reset Button

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 (Lab 4), pop-up box (Lab 5), string name (Lab 6), and numeric hex value (Lab 7) to your driver's form menu:

Step	Action
1	Update the MyWizardDriver.vfr file
2	<p>Add the following code (as shown below after the "GUID" definition Apprx. Line 29):</p> <p>With this code you are created a VFR sub-function called "MyStandardDefault"</p> <pre>defaultstore MyStandardDefault, prompt = STRING_TOKEN(STR_STANDARD_DEFAULT_PROMPT), attribute = 0x0000; // Default ID: 0000 standard default</pre> <pre>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</pre>
3	<p>Add the following code before the "endform" (as shown below Apprx. Line 55):</p> <pre>resetbutton defaultstore = MyStandardDefault, prompt = STRING_TOKEN(STR_STANDARD_DEFAULT_PROMPT_RESET), help = STRING_TOKEN(STR_STANDARD_DEFAULT_HELP), endresetbutton;</pre>

Step	Action
	<pre> 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; </pre>
4	Save MyWizardDriver.vfr
5	Update the MyWizardDriver.uni file
6	<p>Add the following strings at the end of the file to support the “STR_” referenced added in the .vfr file:</p> <pre> #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" </pre>
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	<p>Access the My Wizard Driver menu and notice the item “Reset to Standard Default”</p> 
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	<p>Observe: Notice that since this change requires a reset, the Nt32 will exit out completely.</p> 
25	<p>Press “Enter” to return to the Visual Studio Command Prompt</p> 

For any build issues copy the solution files from C:\Fw\LabSolutions\LessonE.4

NOTE: Del Directory C:\fw\edk2\Build\NT32IA32\DEBUG_VS2010x86\IA32\MyWizardDriver before the Build command to build the MyWizardDriver Clean.

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.






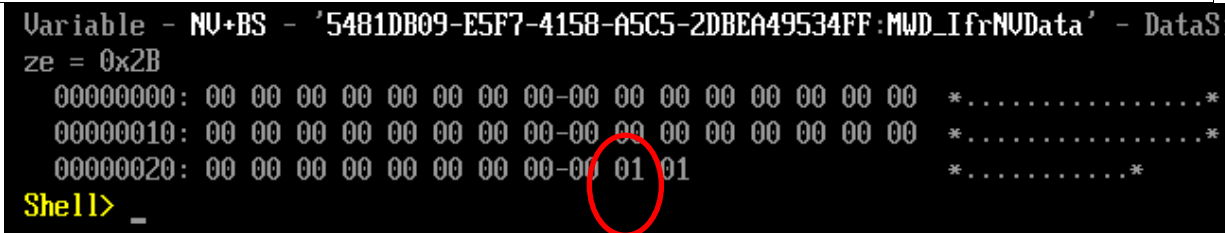
Figure 5 My Wizard Driver with a pop-up box


Step	Action												
	<p><u>Background Information</u> (not a step)</p> <p>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:</p> <table><tr><th>Value</th><th>Display</th><th>String token</th></tr><tr><td>0</td><td>500 Hex</td><td>STR_ONE_OF_TEXT3</td></tr><tr><td>1</td><td>480 Hex</td><td>STR_ONE_OF_TEXT2</td></tr><tr><td>2</td><td>400 Hex</td><td>STR_ONE_OF_TEXT1</td></tr></table> <p>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</p>	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
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											
1	Update the MyWizardDriver.vfr file												
2	Add the following code before the “resetbutton” statement (approximately line 53)												

Step	Action
	<pre> // // 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; </pre>

Step	Action
	<pre> 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 reference 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_OPTION_DEFAULT) 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, </pre>
3	Save the MyWizardDriver.vfr file
4	Update the MyWizardDriver.uni file
5	<p>Add the following code to the end of the file (as shown below):</p> <pre> #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" </pre> <pre> 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" </pre>
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	<p>Test the “grayoutif” by selecting “Enable My XYZ Device” then press the “Space” bar to toggle off or “Disabled”.</p> 
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: <code>dmpstore -all</code>
28	
28	<pre> 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; </pre> <p>File MyWizardDriverNVDataStruc.h</p> <p>By updating MyWizardDriverNVDataStruc.h, our data structure stored in NVRAM is named MWD_IfrNVData of type MYWIZARDDRIVER_CONFIGURATION.</p> <p>Notice that the base address byte is the next to the last byte in the data structure <code>MWD_IfrNVData.MyWizardDriverBaseAddress</code> where 02 == 400H, 01 == 480H, 00 == 500H</p> <p>Notice the NVRAM Variable with the value of 480H will have a true value of 01.</p>

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\LessonE.5

NOTE: Del Directory C:\fw\edk2\Build\NT32IA32\DEBUG_VS2010x86\IA32\MyWizardDriver before the Build command to build the MyWizardDriver Clean.

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.

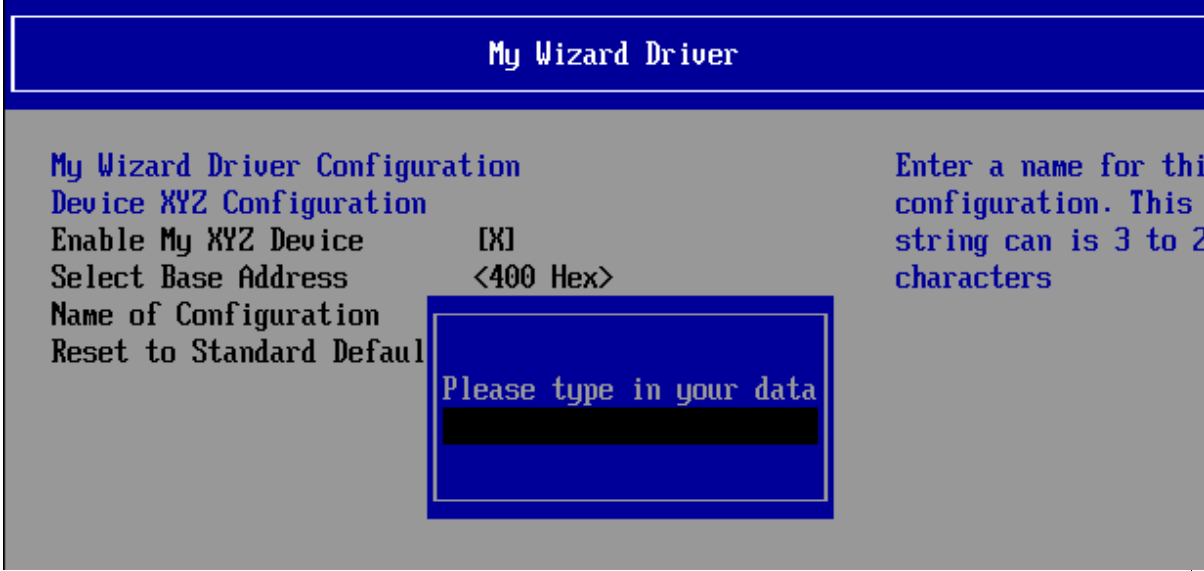
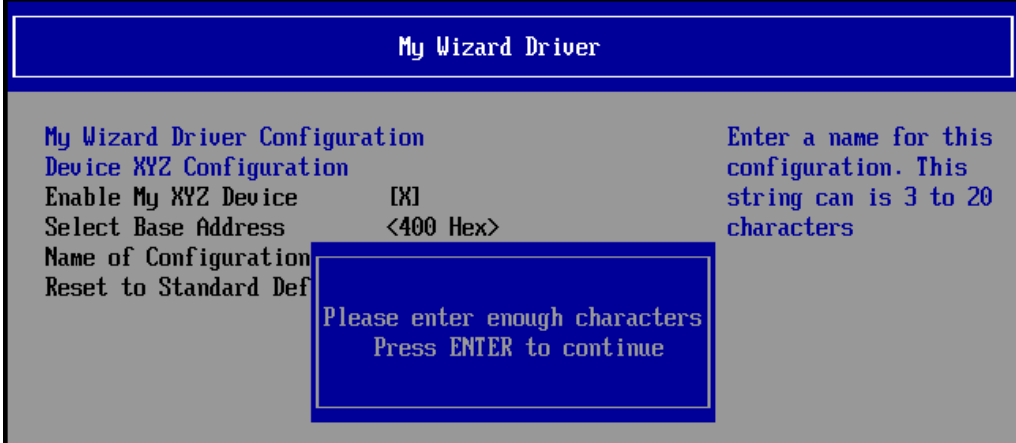


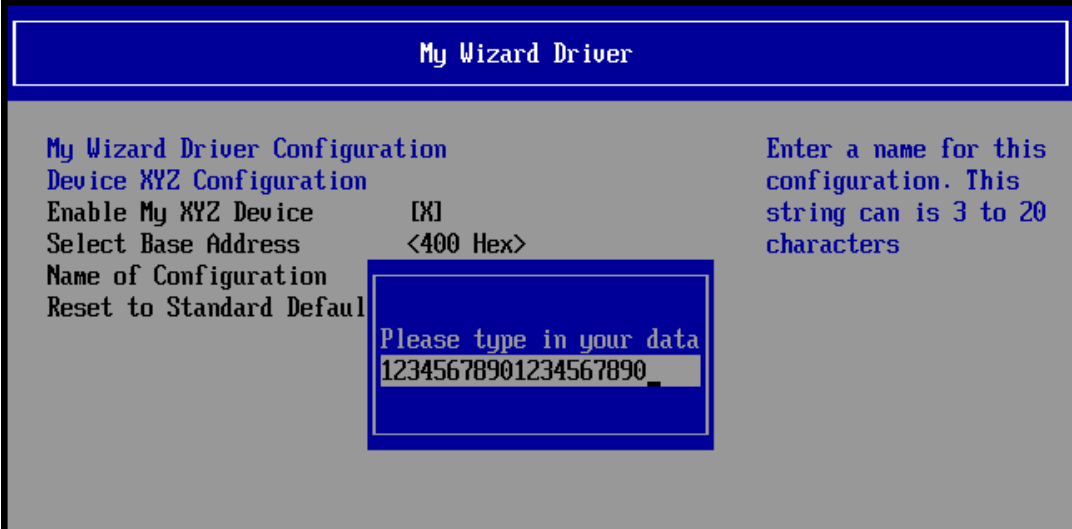

Figure 6 : Menu with a string item


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
	<pre>// // 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;</pre>
	<pre>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</pre>
3	Save MyWizardDriver.vfr
4	Update MyWizardDriver.uni
5	Add the following code to the bottom of the file:

Step	Action
	<pre>#string STR_MY_STRING_PROMPT #language en "Name of Configuration"</pre> <pre>#string STR_MY_STRING_HELP #language en "Enter a name for this configuration. This string can is 3 to 20 characters"</pre> <pre>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" 44 45 #string STR_MY_STRING_PROMPT #language en "Name of Configuration" 46 47 #string STR_MY_STRING_HELP #language en "Enter a name for this configuration. This 48 49</pre>
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	 <p>Notice the string text pop up menu gets displayed</p>
19	Test the “minsize” by only typing your choice of a two character string.
20	<p>Press “Enter”</p>  <p>Notice the an error message validating that you need to enter three or more characters</p>
21	Press “Enter” to clear the message
22	Press “Enter” again to re-enter pop up menu

Step	Action
23	 <p>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.</p>
24	<p>Press “Enter”</p>  <p>Notice that the “Configuration changed” message is displayed</p>
25	<p>Test the grayout if by selecting “Enable My XYZ Device”</p>

Step	Action
26	<p>Press the “Spacebar” to toggle off/disable</p> <p>Notice that the “Select Base Address” and “Name of Configuration” fields are now grayed out and not selectable</p> 
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	<p>At the Shell Prompt, type <code>dmpstore -all</code></p> <p>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</p>

Step	Action
34	<pre> 00000020: 04 00 00 00 00 10 00 00-0E 00 00 00 00 00 00 00 *.....* Variable - NV+BS - '5481DB09-E5F7-4158-A5C5-2DBEA49534FF:MWD_IfrNUData' - DataSi ze = 0x2B 00000000: 31 00 32 00 33 00 34 00-35 00 36 00 37 00 38 00 *1.2.3.4.5.6.7.8.* 00000010: 39 00 30 00 31 00 32 00-33 00 34 00 35 00 36 00 *9.0.1.2.3.4.5.6.* 00000020: 37 00 38 00 39 00 30 00-00 01 01 *7.8.9.0....* Shell> _ </pre> <pre> 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; </pre>
35	Type "reset" at the Shell prompt
36	Press "Enter" to return to the Visual Studio Command Prompt <pre>C:\FW\edk2></pre>

For any build issues copy the solution files from C:\Fw\LabSolutions\LessonE.6

NOTE: Del Directory C:\fw\edk2\Build\NT32IA32\DEBUG_VS2010x86\IA32\MyWizardDriver before the Build command to build the MyWizardDriver Clean.

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	<p>Add the following code in the location shown below at approx. Line 90 and before the “resetbutton” item:</p> <pre>// // 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,</pre>

```

        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"

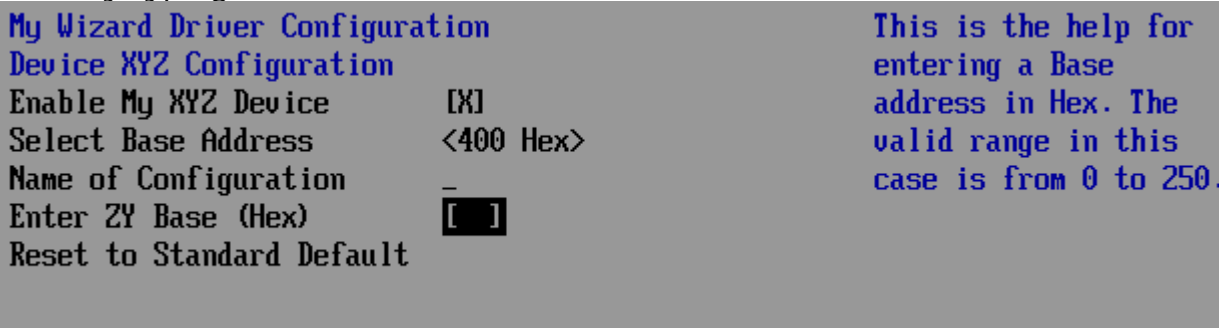

9 Type build run



10 Press "Enter"

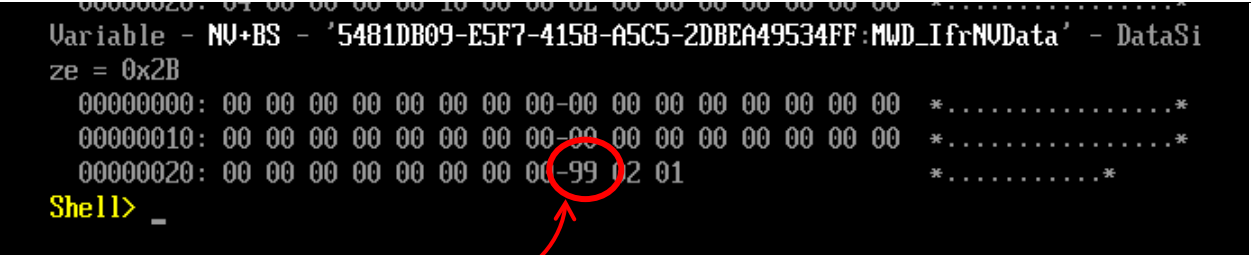

11 Type exit

12 Press "Enter"

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

14	Press “Enter”
15	<p>Inside the Device Manager menu, select “My Wizard Driver Sample Formset”</p> <p>Notice the value for “Enter ZY Base(Hex)” is 022. Hex is the default because of the VFR field “default = 0x22”</p>
16	Select “Enter ZY Base(Hex)”
17	Press “Enter”
18	<p>Test by typing a “M” character</p> 
18	<p>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.</p> <p>The string “!!” is part of the Browser engine :</p> <pre>MdeModulePkg\Universal\SetupBrowserDxe\SetupBrowserStr.uni #string INPUT_ERROR_MESSAGE #language en-US "!!"</pre> 
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	<p>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.</p> 
24	<p>Press "Space bar" again to "Enable My XYZ Device" and the "Enter ZY Base(Hex)" is displayed again</p> 
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	 <pre> 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 00-99 02 01 *.....* Shell> _ </pre> <pre> 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; </pre> <p>Notice by modifying MyWizardDriverNVDataStruc.h our data structure stored in NVRAM is named MWD_IfrNVData of type MYWIZARDDRIVER_CONFIGURATION.</p> <p>Notice that hex data is after the string data at the 21st byte in the data structure MWD_IfrNVData.MyWizardDriverHexData</p>
31	Type “reset” at the Shell prompt
32	Press “Enter” to return to the Visual Studio Command Prompt 

For any build issues copy the solution files from C:\Fw\LabSolutions\LessonE.7

NOTE: Del Directory C:\fw\edk2\Build\NT32IA32\DEBUG_VS2010x86\IA32\MyWizardDriver before the Build command to build the MyWizardDriver Clean.

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)

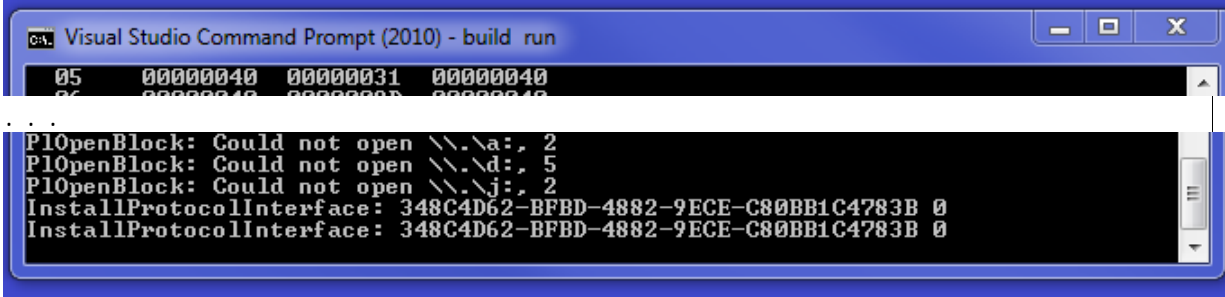

```
DEBUG ((DEBUG_INFO, "\n:: START Call back ,Question ID=0x%08x Type=0x%04x
Action=0x%04x", QuestionId, Type, Action));
```

a. Add the Case statements to the Call back routine

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

Step	Action
	<pre> 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 </pre>

Step	Action
	<pre> 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 } </pre>
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
	

b. 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,

	<pre> 76 77 // 78 // Define a string (EFI_IFR_STRING) to name the configuration in the 79 // NVRAM variable 80 // 81 grayoutif ideqval MWD_IfrNVDData.MyWizardDriverChooseToEnable == 0x0; 82 string varid = MWD_IfrNVDData.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 </pre>	
3	<p>Include the numeric item by adding the following code in the location shown below, Approx. line 97 and line 100</p>	
	<pre> questionid = 0x1111, INTERACTIVE </pre>	
	<pre> 92 // 93 // Define a numeric free form menu item 94 // 95 suppressif ideqval MWD_IfrNVDData.MyWizardDriverChooseToEnable == 0x0; 96 numeric varid = MWD_IfrNVDData.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; 107 </pre>	
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 question ID 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=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=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=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\LessonE.8

NOTE: Del Directory C:\fw\edk2\Build\NT32IA32\DEBUG_VS2010x86\IA32\MyWizardDriver before the Build command to build the MyWizardDriver Clean.

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

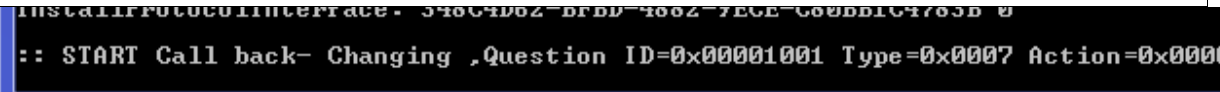
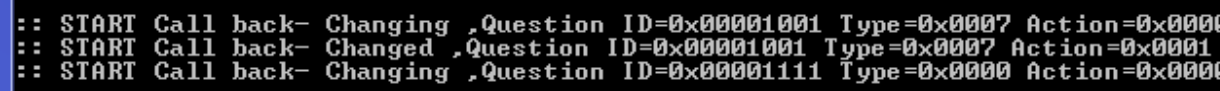
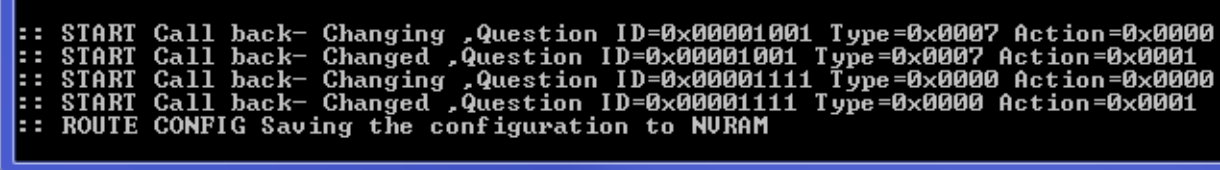

In this lab, you'll update your driver to print debug statements when the Hii browser engine calls back into your call back function. Every time the browser does anything with the interactive labeled fields there is a call made to your driver's call back function. We can determine the item by the questionid 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	<p>Comment out the DEBUG statement with <code>/// MyWizardDriverHiiConfigAccessCallback</code> 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</p> <pre>// 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</pre>
3	<p>Add a switch case statement of the question ID's to the "Action" switch case of <code>EFI_BROWSER_ACTION_CHANGING</code> in the call back function by adding a nested switch case code (as shown below at approx. line 372)</p>

Step	Action
	<pre> 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; } </pre> <pre> 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 </pre>
4	<p>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):</p>

Step	Action
	<pre> 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; } </pre>
	<pre> 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 </pre>
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	 Notice: when changing the “ Name of Configuration ” field
18	 Notice: when changing the “ Enter ZY Base(Hex) ” field
19	 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 

For any build issues copy the solution files from C:\Fw\LabSolutions\LessonE.9

NOTE: Del Directory C:\fw\edk2\Build\NT32IA32\DEBUG_VS2010x86\IA32\MyWizardDriver before the Build command to build the MyWizardDriver Clean.

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).

MyWizardDriver Second Setup Page		
Windows System Date	[03/07/2013]	This is the help for the Date (month/day/year). (Error checking will be done against month/day/year combinations that are not supported.)
Windows System Time	[16:42:05]	
My System Date	[01/01/2013]	
My System Time	[15:33:33]	
► Enter Page 1		
+/- =Adjust Value F9=Reset to Defaults F10=Save ↑↓+=Move Highlight <Enter>=Select Entry Esc=Exit		

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): <pre> EFI_HII_TIME Time; EFI_HII_DATE Date; </pre>

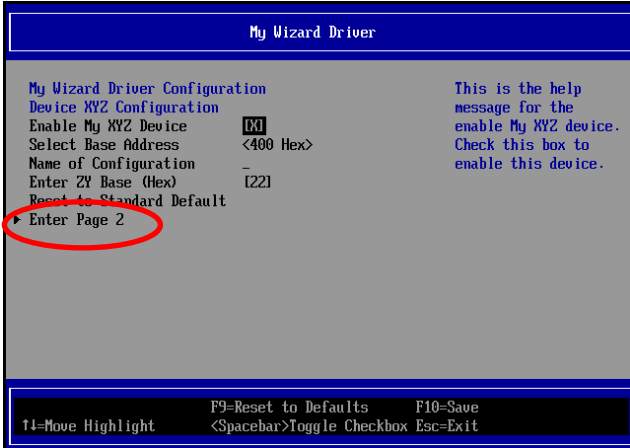
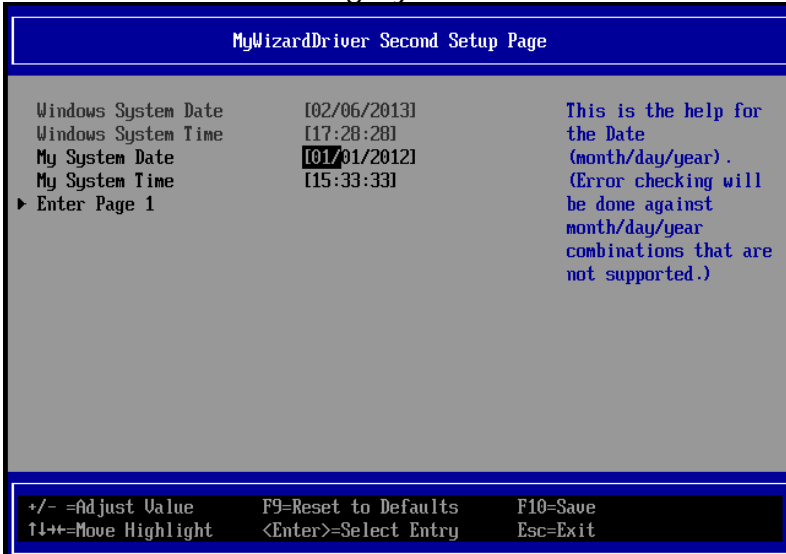
Step	Action
	<pre> 21 #pragma pack(1) 22 typedef struct { 23 24 UINT16 MyWizardDriverStringData[20]; 25 UINT8 MyWizardDriverHexData; 26 UINT8 MyWizardDriverBaseAddress; 27 UINT8 MyWizardDriverChooseToEnable; 28 EFI_HII_TIME Time; 29 EFI_HII_DATE Date; 30 } MYWIZARDDRIVER_CONFIGURATION; 31 </pre>
3	Save MyWizardDriverNVDataStruc.h
4	Update the MyWizardDriver.uni file
5	<p>Add the following code to the end of the file to update the second page's string:</p> <pre> #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" </pre>
6	Save MyWizardDriver.uni
7	Update the MyWizardDriver.vfr file
8	<p>Add the "goto" VFR item to allow browser to ender another form by adding the following code before the "endform" at approx. line 114</p> <pre> grayoutif idequal MWD_IfrNVData.MyWizardDriverChooseToEnable == 0x0; goto 2, prompt = STRING_TOKEN(STR_GOTO_FORM2), //SecondSetupPage help = STRING_TOKEN(STR_GOTO_HELP); endif; </pre>


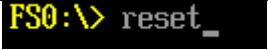

Step	Action
	<pre> 108 resetbutton 109 defaultstore = MyStandardDefault, 110 prompt = STRING_TOKEN(STR_STANDARD_DEFAULT_PROMPT_RESET), 111 help = STRING_TOKEN(STR_STANDARD_DEFAULT_HELP), 112 endresetbutton; 113 114 grayoutif ideqval MWD_IfrNVData.MyWizardDriverChooseToEnable == 0x0; 115 goto 2, 116 prompt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

Step	Action
	<pre> 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 </pre>

Step	Action
	<pre> 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 </pre>

Step	Action
	<pre> 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; </pre>
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	<p>Press “Enter”</p> <p>Notice the “Enter Page 2” option. Without goto in the MyWizardDriver.vfr file, you wouldn’t be able to access page two.</p> 
20	Select “Enter Page 2”
21	<p>Press “Enter”</p> <p>Notice how the Windows System Date and Time cannot be modified to any other date/time and is grayed out:</p> 
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


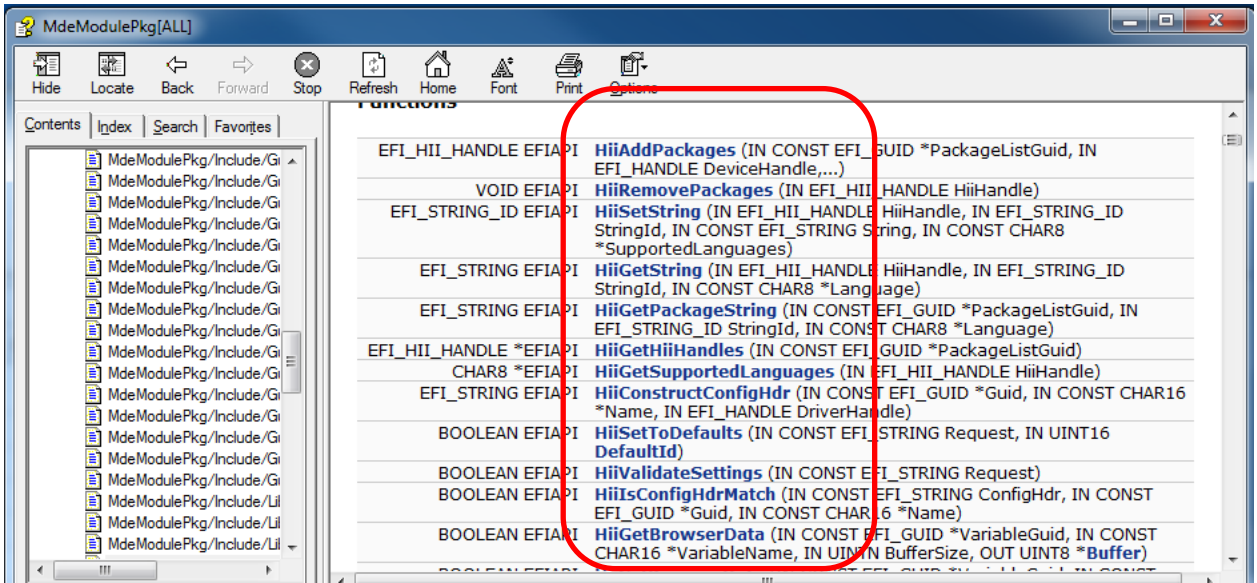
Step	Action
26	<p>Notice the “Select Base Address” , “Name of Configuration” and the “Enter Page 2” fields are now grayed out and not selectable</p>  <p>Note: If you wanted to hide “Enter Page 2”, you would simply replace the <code>grayoutif</code> code you entered with <code>suppressif</code> in the VFR file</p>
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	<p>Press “Enter” to return to the Visual Studio Command Prompt</p> 

For any build issues copy the solution files from C:\Fw\LabSolutions\LessonE.10

NOTE: Del Directory C:\fw\edk2\Build\NT32IA32\DEBUG_VS2010x86\IA32\MyWizardDriver before the Build command to build the MyWizardDriver Clean.


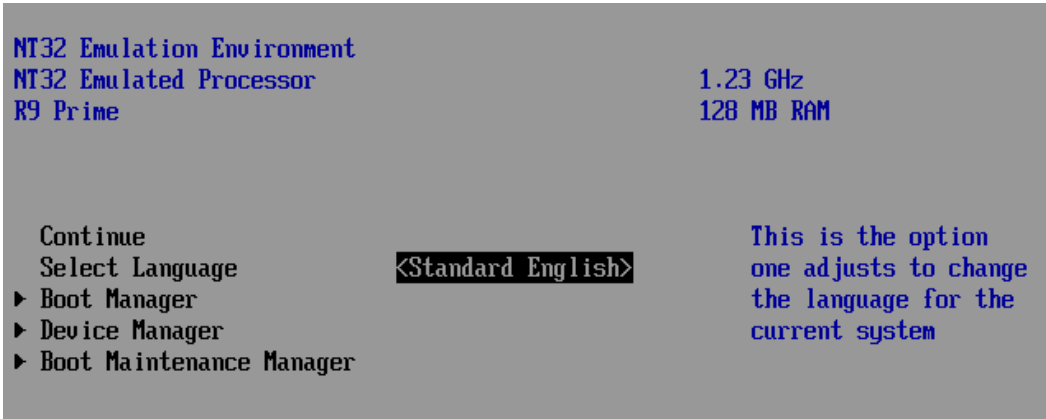


11. Adding Communication from Driver to Console through HII

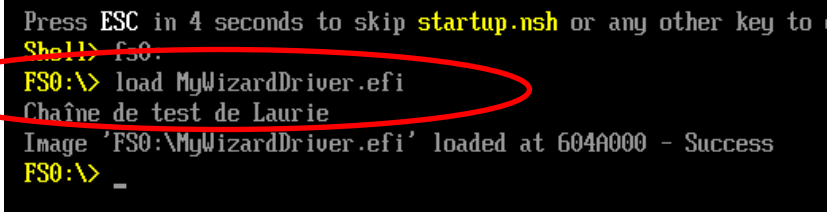


In this lab, you'll add communication from the driver to the console through HII. More specifically, you'll add code to retrieve a string from the HII database and print the string to the console. Then, you'll add the string in French, change the language, and test to ensure the correct language is displayed. 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	 <p>Note: Notice the list of Hii function calls available. To get Strings the HiiGetString function can be used.</p>

Step	Action
5	Update the C:\Fw\edk2\Nt32pkg\Nt32pkg.fdf file
6	<p>Make your driver stand alone. Remove (or comment out) the include statement in the Nt32pkg.fdf file:</p> <pre>#INF MyWizardDriver/MyWizardDriver.inf</pre> <pre>INF MdeModulePkg/Universal/Network/IScsiD #INF MyWizardDriver/MyWizardDriver.inf !if \$(BUILD_NEW_SHELL) == TRUE INF ShellPkg/Application/Shell/Shell.inf !endif</pre>
7	Save Nt32pkg.fdf
8	Update the MyWizardDriver.uni file
9	<p>Add the following code to the top of the file at approx. line 14 as shown:</p> <pre>#langdef fr-FR "Francais"</pre> <pre>12 13 #langdef en "English" 14 #langdef fr-FR "Francais" 15 16 #string STR_SAMPLE_FORM_SET_TITLE #language en "My Wizard I 17 #string STR_SAMPLE_FORM_SET_HELP #language en "Help for S 18 #string STR_SAMPLE_FORM1_TITLE #language en "My Wizard I 19</pre>
10	<p>Add the following code to the end of the file:</p> <pre>#string STR_LANGUAGE_TEST_STRING #language en "Laurie's Test String" #language fr-FR "Chaîne de test de Laurie"</pre> <pre>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</pre>
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):

Step	Action
	<pre>EFI_STRING StringPtr;</pre> <div> <pre> 189 UINTN BufferSize; 190 MYWIZARDDRIVER_CONFIGURATION *Configuration; 191 BOOLEAN ActionFlag; 192 EFI_STRING StringPtr; 193 Status = EFI_SUCCESS; 194 </pre> </div>
14	<p>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):</p> <pre>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);</pre> <div> <pre> 362 FreePool (ConfigRequestHdr); 363 364 StringPtr = HiiGetString (HiiHandle[0], STRING_TOKEN (STR_LANGUAGE_TEST_ST 365 DEBUG ((EFI_D_INFO, "[MyWizardDriver-Entrypoint] My String was: %s\n", StringPtr) 366 Print(L"%s\n", StringPtr); 367 368 // end HII 369 // 370 // Install Driver Supported EFI Version Protocol onto ImageHandle </pre> </div>
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	<p>Type load MyWizardDriver.efi and notice that the string's English version is displayed:</p> <div> <pre> Shell> fs0: FS0:\> load MyWizardDriver.efi Laurie's Test String Image 'FS0:\MyWizardDriver.efi' loaded at 604A000 - Success FS0:\> _ </pre> </div>

Step	Action
23	Type <code>Reset</code> 
24	Press "Enter"
25	Type <code>build</code>
26	Type <code>build run</code>
27	Press "Enter"
28	Type <code>exit</code> at the shell prompt
29	Select Language 
30	Press "Enter"
31	 Select "Français"
32	Press "Enter"
33	 Select "Continuer"
34	Press "Enter"

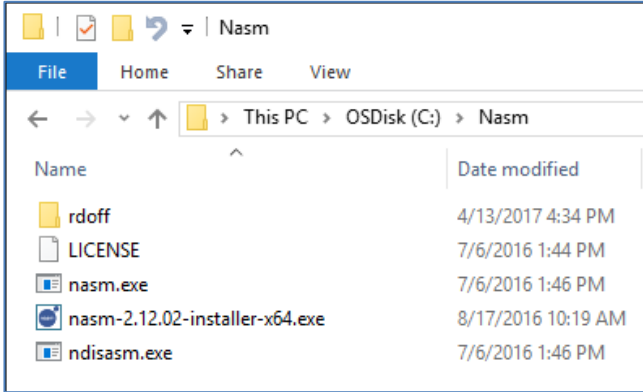
Step	Action
35	At the Shell Prompt, type <code>Fs0:</code>
36	<p>Type <code>load MyWizardDriver.efi</code></p>  <pre> Press ESC in 4 seconds to skip startup.nsh or any other key to Shell> fs0: FS0:\> load MyWizardDriver.efi Chaîne de test de Laurie Image 'FS0:\MyWizardDriver.efi' loaded at 604A000 - Success FS0:\> _ </pre> <p>Notice that the string's French version is displayed:</p>
37	Type <code>"reset"</code> 
38	<p>Press "Enter" to return to the Visual Studio Command Prompt</p> 

For any build issues copy the solution files from C:\Fw\LabSolutions\LessonE.11


NOTE: Del Directory C:\fw\edk2\Build\NT32IA32\DEBUG_VS2010x86\IA32\MyWizardDriver before the Build command to build the MyWizardDriver Clean.

Make sure you update Nt32Pkg.fdf.

Lab Setup

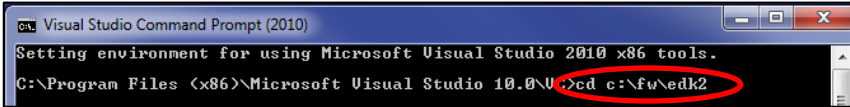
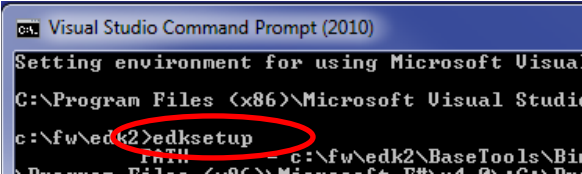
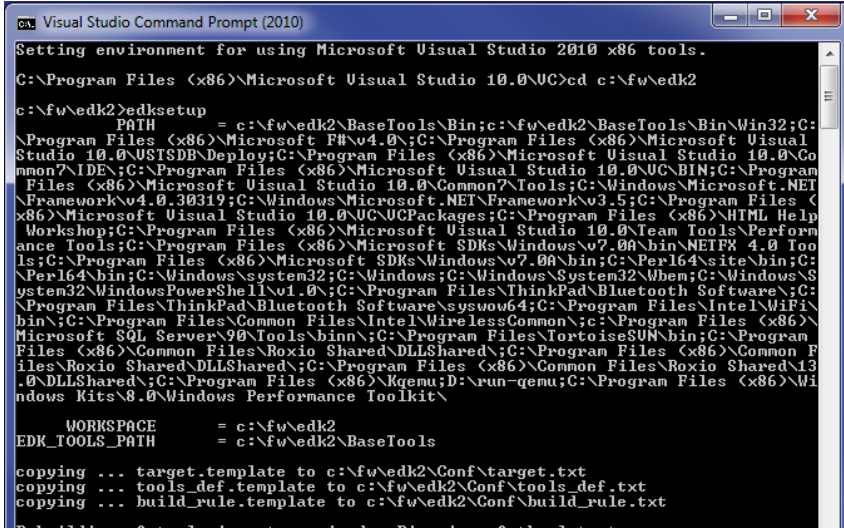
Step	Action
28	Download the UEFI Training Materials (accept any security notifications)
29	Click “Open”
30	<p>Unzip the file to C: which will take a few minutes</p> <ul style="list-style-type: none"> C:\Fw\Presentations - separate zip file C:\Fw\Edk2 – Open source tianocore.org EDK II C:\Fw\DriverWizard – Install .MSI C:\Fw\LabSampleCode - Solutions for Labs C:\Fw\Documentation - .chm files and examples C:\Fw\Nasm – For Assembly compiler <p>Note: It is highly important that you unzip the file correctly to this location because all the file locations in this training guide follow that format.</p>
31	<p>Copy the C:\fw\NASM directory to C:</p> 

Pin Visual Studio Command Prompt for Windows

Step	Action
1	Visual Studio Command Prompt for Windows 10
2	<p>Pin “Visual Studio Command Prompt (201n)” to the Task bar</p>  <p>Note: this is the prompt that will be used to build the EDK II Labs</p>

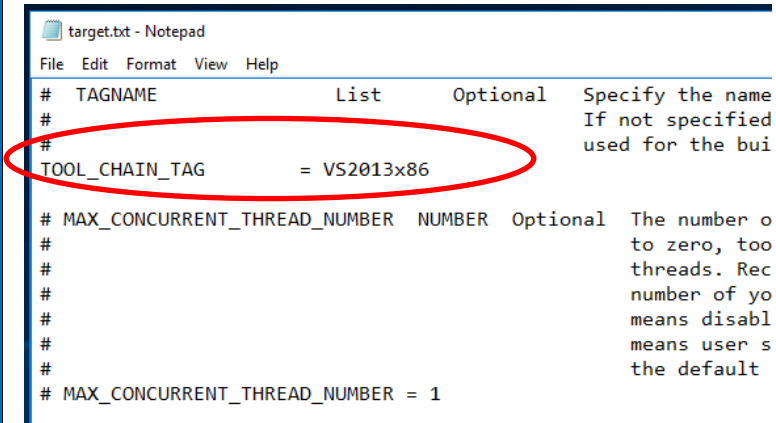
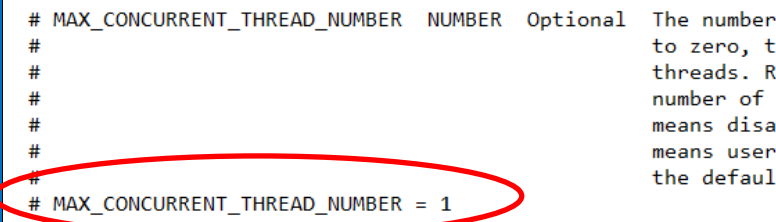
Preparing for the BUILD Command

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


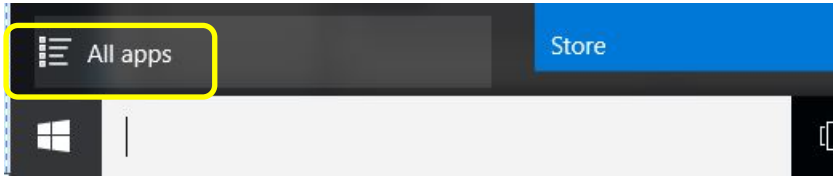
Step	Action
4	Open Visual Studio Command Prompt
32	Type CD c:\fw\edk2 
33	Press “Enter”
34	Type edksetup 
35	Press “Enter”  <p>Note: If you see “!!! WARNING !!!...”, don’t be alarmed. That can be ignored at this time.</p>

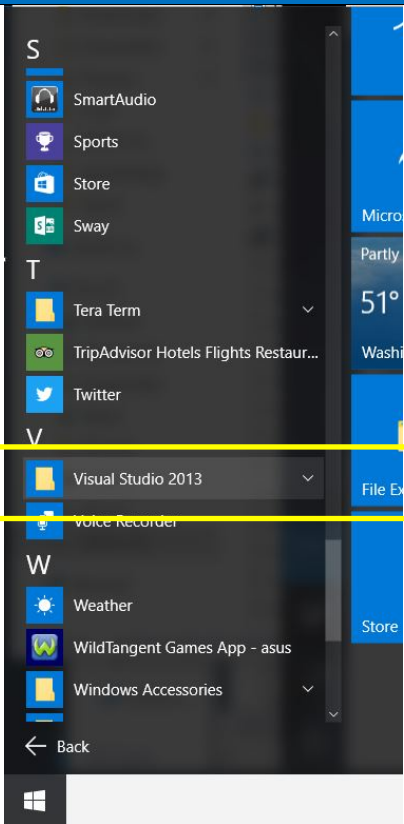
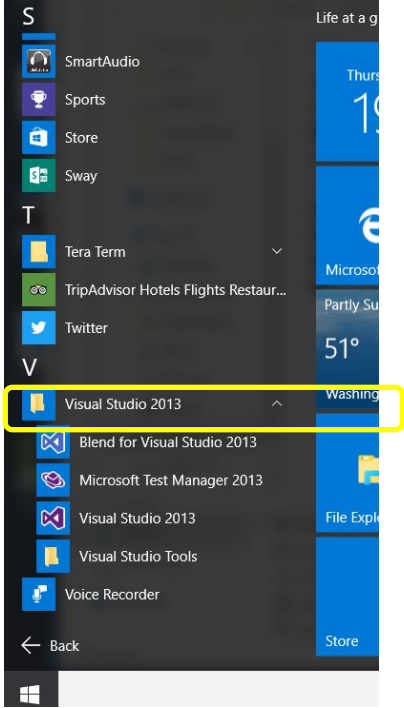
Configuring Build Tools


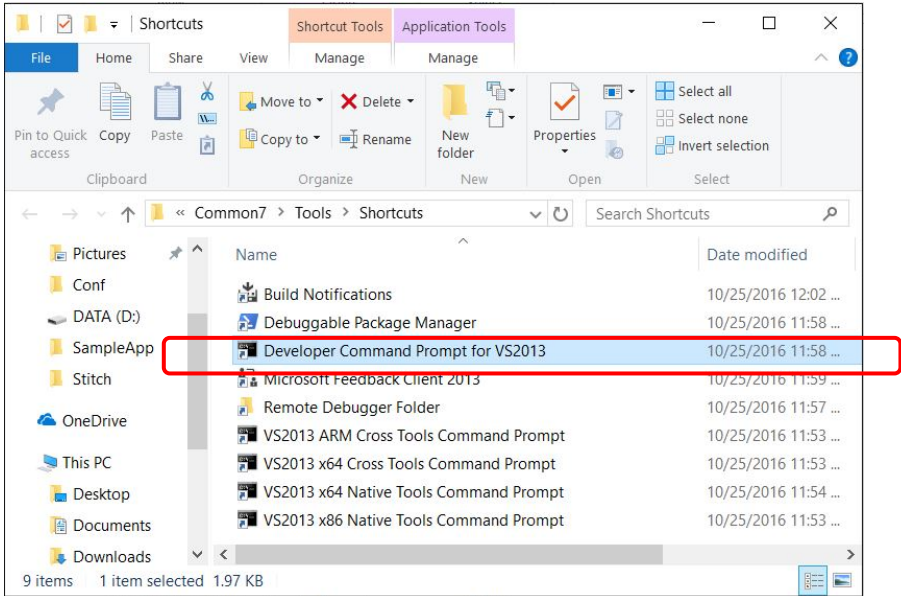
Note: You only need to edit *Target.txt* and/or *Tools_Def.txt* once after the first **edksetup** command after downloading the Training materials .zip file.

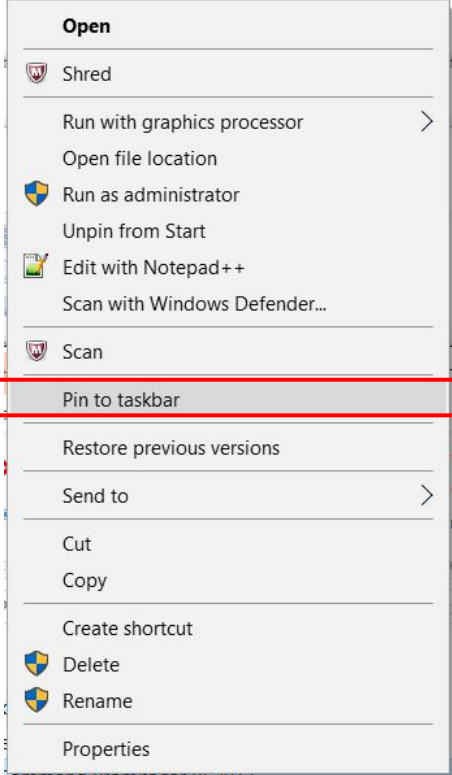

Step	Action
36	Open Notepad or other text editor that supports UNICODE
37	Open C:/fw/edk2/Conf/Target.txt
38	<p>Use the Microsoft Windows and Visual Studio Matrix</p> <p>File: C:/fw/edk2/conf/target.txt</p> <p>Modify TOOL_CHAIN_TAG to match your version of Visual Studio.</p>  <pre> # TAGNAME List Optional Specify the name # If not specified # used for the bui # # TOOL_CHAIN_TAG = VS2013x86 # # MAX_CONCURRENT_THREAD_NUMBER NUMBER Optional The number o # to zero, too # threads. Rec # number of yo # means disabl # means user s # the default # # MAX_CONCURRENT_THREAD_NUMBER = 1 </pre> <p>Example:</p> <p>for Windows 7 64 bit OS and Visual Studio 2013 modify the following in Target.txt</p> <p>From:</p> <pre>TOOL_CHAIN_TAG = MYTOOLS</pre> <p>to:</p> <pre>TOOL_CHAIN_TAG = VS2013x86</pre>
39	<p>OPTIONAL: Update the MAX_CONCURRENT_THREAD_NUMBER</p> <p>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.</p>  <pre> # MAX_CONCURRENT_THREAD_NUMBER NUMBER Optional The number # to zero, t # threads. R # number of # means disa # means user # the default # # MAX_CONCURRENT_THREAD_NUMBER = 1 </pre>
40	Save and close the text file C:/fw/edk2/conf/target.txt

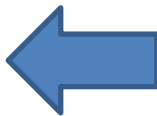
Microsoft Windows 10 – Visual Studio command prompt

Step	Action
	Open Microsoft Visual Studio 2013 command prompt (NOTE: VS 2013 is used in the following examples but VS 2010, VS 2015 and VS 2017 will be similar)
1	Using the Start menu in Windows 10, Left Click on “Windows Key” Lower Left 
2	Left-click on “All apps” Windows 8 (Window 10 use scroll bar next step below) 
3	Scroll down from the scroll bar on the right until “ Visual Studio 2013 ”

Step	Action
	
4	<p>Left Click “Visual Studio 201n”</p> 

Step	Action
5	<p>Left Click “Visual Studio Tools”</p> 
6	<p>Another window will open</p>  <p>Right Click on “Developer Command Prompt for VS201n” NOTE: Make sure you are selecting the correct Command prompt for Visual Studio. For VS2013 it is: <i>“Developer Command Prompt for VS2013”</i></p>
7	<p>Left Click on “Pin to taskbar”</p>

Step	Action
	 <p>A screenshot of a Windows context menu. The 'Pin to taskbar' option is highlighted with a red rectangular box. Other visible options include 'Open', 'Shred', 'Run with graphics processor', 'Open file location', 'Run as administrator', 'Unpin from Start', 'Edit with Notepad++', 'Scan with Windows Defender...', 'Scan', 'Restore previous versions', 'Send to', 'Cut', 'Copy', 'Create shortcut', 'Delete', 'Rename', and 'Properties'.</p>
8	 <p>A screenshot of a Windows taskbar. The 'Developer Command Prompt for VS...' icon is pinned to the taskbar and highlighted with a red rectangular box. The 'Unpin from taskbar' option is also visible in the context menu. The taskbar includes the Start button, search bar, and several application icons.</p> <p>Note: this is the prompt that will be used to build the EDK 2 Labs</p>

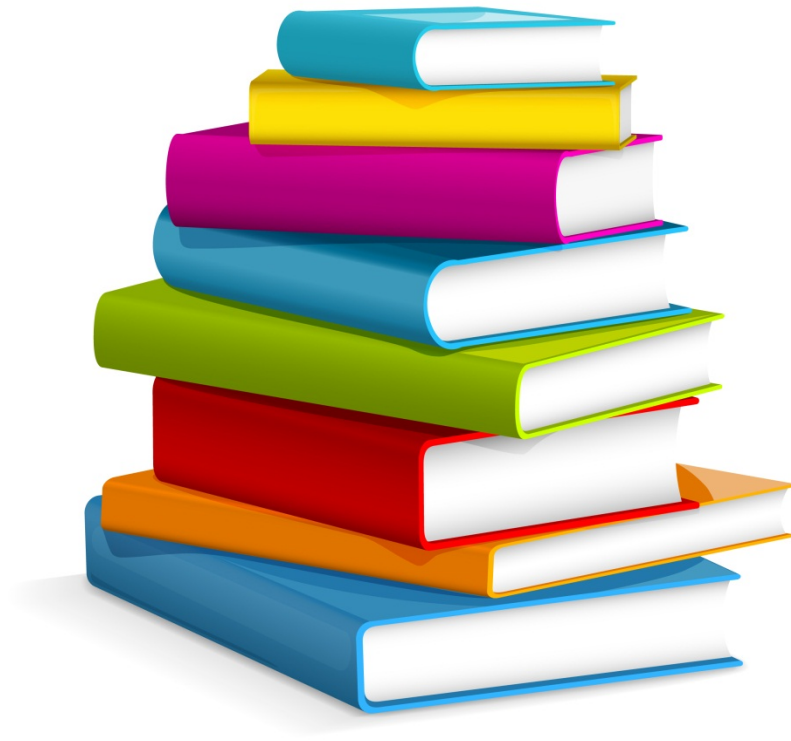


Microsoft Windows and Visual Studio Matrix

Files `C:/fw/edk2/conf/target.txt` and `tools_def.txt` ([Return to Configuring Build Tools](#))

Visual Studio	Version	Win7,Win8 & Win10 x64
VS2005	8.0	<code>target.txt</code> TOOL_CHAIN_TAG = VS2005x86 Requires WinDDK Link16**
VS2008	9.0	<code>target.txt</code> TOOL_CHAIN_TAG = VS2008x86
VS2010	10.0	<code>target.txt</code> TOOL_CHAIN_TAG = VS2010x86
VS2012	11.0	<code>target.txt</code> TOOL_CHAIN_TAG = VS2012x86
VS2013	12.0	<code>target.txt</code> TOOL_CHAIN_TAG = VS2013x86
VS2015	14.0	<code>target.txt</code> TOOL_CHAIN_TAG = VS2015x86
VS2017	15.0	<code>target.txt</code> TOOL_CHAIN_TAG = VS2017x86

Reference



Glossary of UEFI Terms and Acronyms

<https://github.com/tianocore/tianocore.github.io/wiki/Acronyms-and-Glossary>

Helpful Links

UEFI Forum	http://www.uefi.org
UEFI Open Source	http://www.tianocore.org
EFI Developer Kit (EDK II)	http://tianocore.github.io/edk2.html
EDK II Documents	http://www.tianocore.org/docs/
UEFI Shell Documents	https://github.com/tianocore/tianocore.github.io/wiki/ShellPkg

Acknowledgement

- Redistribution and use in source (original document form) and 'compiled' forms (converted to PDF, epub, HTML and other formats) with or without modification, are permitted provided that the following conditions are met:
- Redistributions of source code (original document form) must retain the above copyright notice, this list of conditions and the following disclaimer as the first lines of this file unmodified.
- Redistributions in compiled form (transformed to other DTDs, converted to PDF, epub, HTML and other formats) must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- THIS DOCUMENTATION IS PROVIDED BY TIANOCORE PROJECT "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL TIANOCORE PROJECT BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS DOCUMENTATION, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
- Copyright (c) 2018, Intel Corporation. All rights reserved.