TABLE OF CONTENTS

UEFI DRIVER HII LINUX LAB GUIDE

UEFI and EDK II Base Training

Lab and Reference Guide

Assumes Ubuntu 16.04

DRAFT FOR REVIEW

11/08/2018 07:59:16

Revision 01.0

Contributed by

Laurie Jarlstrom, Intel Corporation

Acknowledgements

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.

Revision History

Revision	Revision History	Date
01.0	Initial release.	July 2018

UEFI DRIVER WIZARD – ADDING HII

Lab 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.

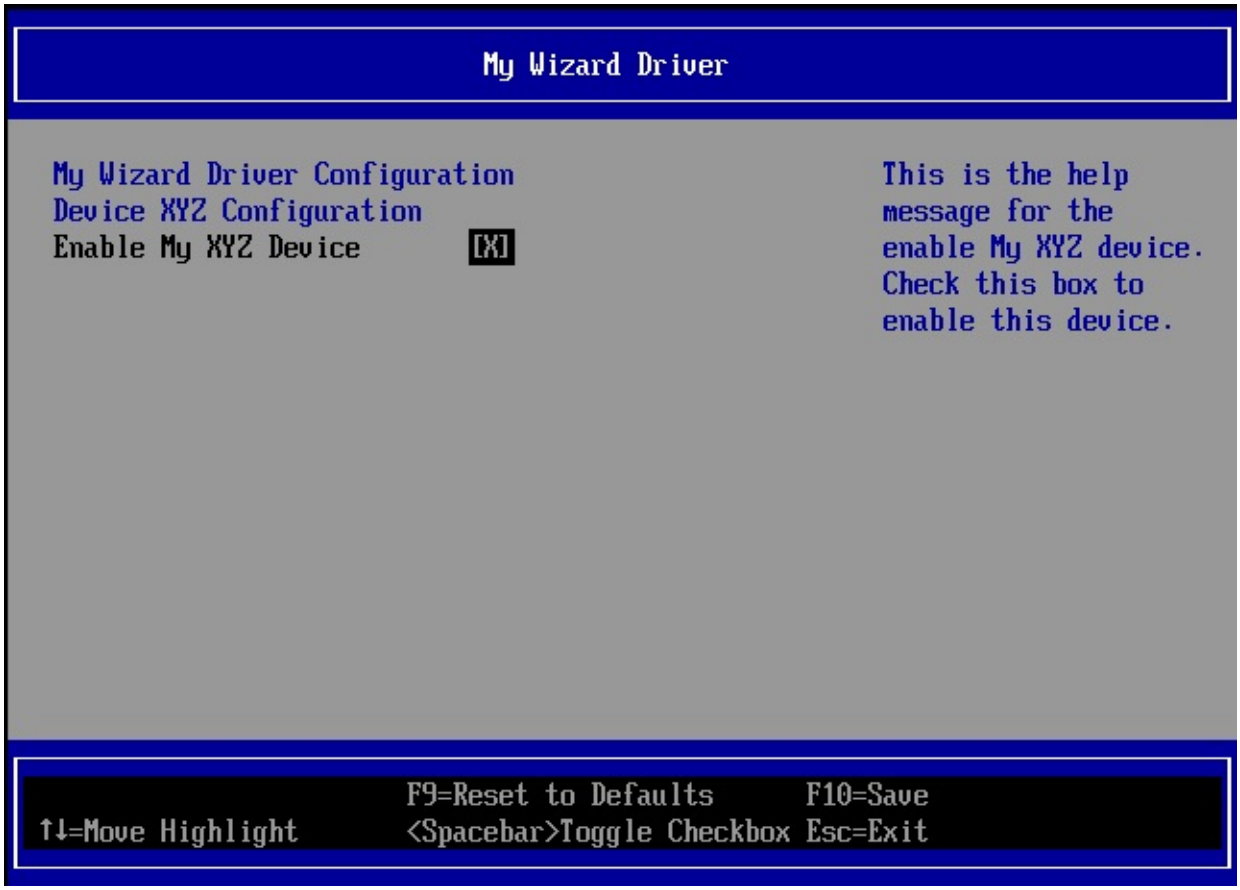


Figure 1 My Wizard Driver menu with strings and forms

Lab 1 a. Setup for Lab adding HII

Complete Lab Setup to configure for building with OvmfPkg with QEMU.

Skip to Lab 1.b if the UEFI Driver Porting lab was completed.

1). Start with LAB 6. on Driver porting Lab solution and **create** a folder called **MyWizardDriver** in the ~/src/edk2 workspace
2). Now**, locate** and **open**: ~/SRC/LabSampleCode\MyWizardDriver
3). **Copy** the following Files to ~/SRC/edk2/MyWizardDriver

- ComponentName.c ComponentName.h
- DriverBinding.h
- HiiConfigAccess.c HiiConfigAccess.h
- MyWizardDriver.c MyWizardDriver.h
- MyWizardDriver.inf MyWizardDriver.uni
- MyWizardDriver.vfr MyWizardDriverNVDataStruc.h
- SimpleTextOutput.c SimpleTextOutput.h

4). **Update** src/edk2/OvmfPkg/OvmfPkgX64.dsc and add the MyWizardDriver.inf to `[Components]` section:
Hint: add to the last module in the `[Components]` section

```
  MyWizardDriver/MyWizardDriver.inf{
    <LibraryClasses>    DebugLib|MdePkg/Library/UefiDebugLibConOut/UefiDebugLibConOut.i
nf
  }
```

5). **Save** and **close** the file

6). Open Terminal Command Prompt
`bash$ cd ~/src/edk2`

Lab 1 b. Edit Driver for adding HII

1). **Open** the following files for updating:

- MyWizardDriverNVDataStruc.h
- MyWizardDriver.vfr
- MyWizardDriver.uni
- MyWizardDriver.h
- MyWizardDriver.c
- MyWizardDriver.inf

2). **Update** the MyWizardDriverNVDataStruc.h file by copying and pasting the following GUID as shown below: This GUID is used to communicate to the HII Database and Browser Engine

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

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

3). **Save** MyWizardDriverNVDataStruc.h

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

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

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

6). Continue **adding** the remaining code to MyWizardDriver.vfr. This is a Enable/ Disable question for the setup menu in the form of a Check box.

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

    subtitle text = STRING_TOKEN(STR_SUBTITLE_TEXT2);
  //
  // Define a checkbox to enable / disable the device
  //
      checkbox varid   = MWD_IfrNVData.MyWizardDriverChooseToEnable,
                 prompt   = STRING_TOKEN(STR_CHECK_BOX_PROMPT),
                 help     = STRING_TOKEN(STR_CHECK_BOX_HELP),
                 //
                 // CHECKBOX_DEFAULT indicate this checkbox is marked
               //   with EFI_IFR_CHECKBOX_DEFAULT
                 //
                 flags    = CHECKBOX_DEFAULT ,
                 key      = 0,
                 default  = 1,
     endcheckbox;

   endform;

endformset;
```

7). **Save** MyWizardDriver.vfr

8). **Update** 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 Forms
et"
  #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 th
e enable My XYZ device. Check this box to enable this device."
```

9). **Save** MyWizardDriver.uni
10). **Update** the MyWizardDriver.h file. **Add** the following HII libraries starting at approximately line 41 (as shown below) by copying and pasting:

```
// 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:

```
   #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, MY
WIZARDDRIVER_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, Config
102
103 #pragma pack(1)
104 ///
105 /// HII specific Vendor Device Path definition.
106 ///
107 typedef struct {
108   VENDOR_DEVICE_PATH              VendorDevicePath;
109   EFI_DEVICE_PATH_PROTOCOL        End;
110 } HII_VENDOR_DEVICE_PATH;
111
112 #pragma pack()
113 //
114 // Include files with function prototypes
```

12). **Save** MyWizardDriver.h

13). **Update** 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.

```
// Begin code
//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)
    }
  }
};
// end code
```

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):

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

```
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**

```
//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];
// end code
```

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

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

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   //
```

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

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):

```
  // Begin code
  PrivateData->HiiHandle[0] = HiiHandle[0];

   BufferSize = sizeof (MYWIZARDDRIVER_CONFIGURATION);

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

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

18). **Save** MyWizardDriver.c

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

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

```
48 [Protocols]
49   gEfiDriverBindingProtocolGuid
50   gEfiPciIoProtocolGuid
51   gEfiDriverSupportedEfiVersionProtocolGuid
52   gEfiHiiPackageListProtocolGuid
53   gEfiHiiDatabaseProtocolGuid
54   gEfiComponentName2ProtocolGuid
55   gEfiComponentNameProtocolGuid
56   gEfiHiiConfigAccessProtocolGuid
57   gEfiSimpleTextOutProtocolGuid
58
59
60   gEfiHiiStringProtocolGuid
61   gEfiHiiConfigRoutingProtocolGuid
62   gEfiFormBrowser2ProtocolGuid
63   gEfiHiiDatabaseProtocolGuid
64
```

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

21). **Add** MyWizardDriver.inf to the OvmfPkgX64.dsc(See Lab 2 building MyWizardDriver from the Driver Porting Lab)

Build and test MyWizardDriver

1. At the Terminal Command Prompt (**Cntl-Alt-T**)

```
bash$ cd ~/src/edk2
bash$ build
```

2. **Copy** MyWizardDriver.efi to hda-contents

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

3. **Invoke** Qemu

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

4. At the UEFI Shell prompt,**type** fs0:
5. **Type** Load MyWizardDriver.efi and **Press** “Enter”
   This will load your driver into memory

```
FS0:\> load MyWizardDriver.efi
Image 'FS0:\MyWizardDriver.efi' loaded at 5EB9000 - Success
FS0:\> exit_
```

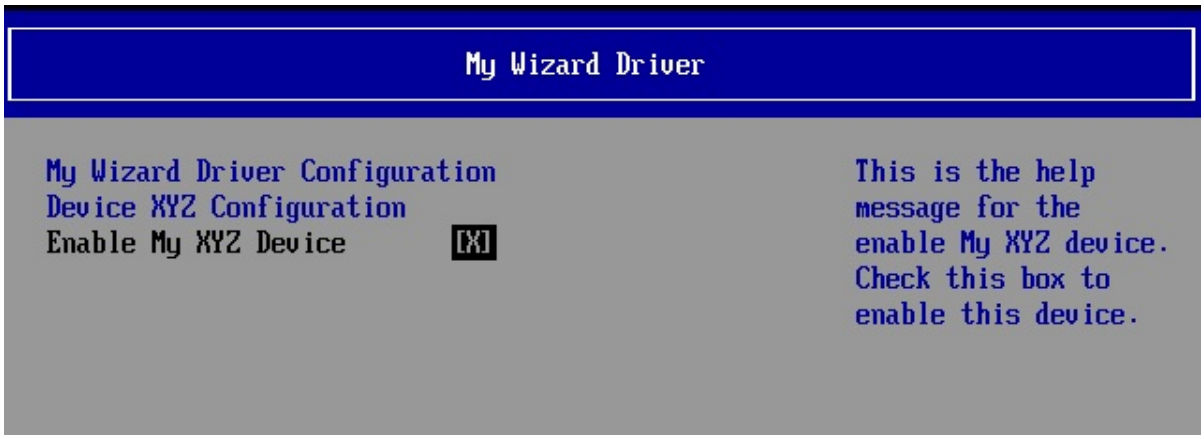
6. **Type** exit
7. Now at the setup front page menu, **select** “Device Manager”

```
  Continue                                    This selection will
  Select Language          <English>          take you to the
▶ Boot Manager                               Device Manager
▶ Device Manager
▶ Boot Maintenance Manager
```

8. **Press** “Enter” to enter **“Device Manager”**
9. **Inside the Device Manager menu** press **the down to “My Wizard Driver Sample Formset”** Press** "Enter"

```
                        Device Manager

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

  Press ESC to exit
```

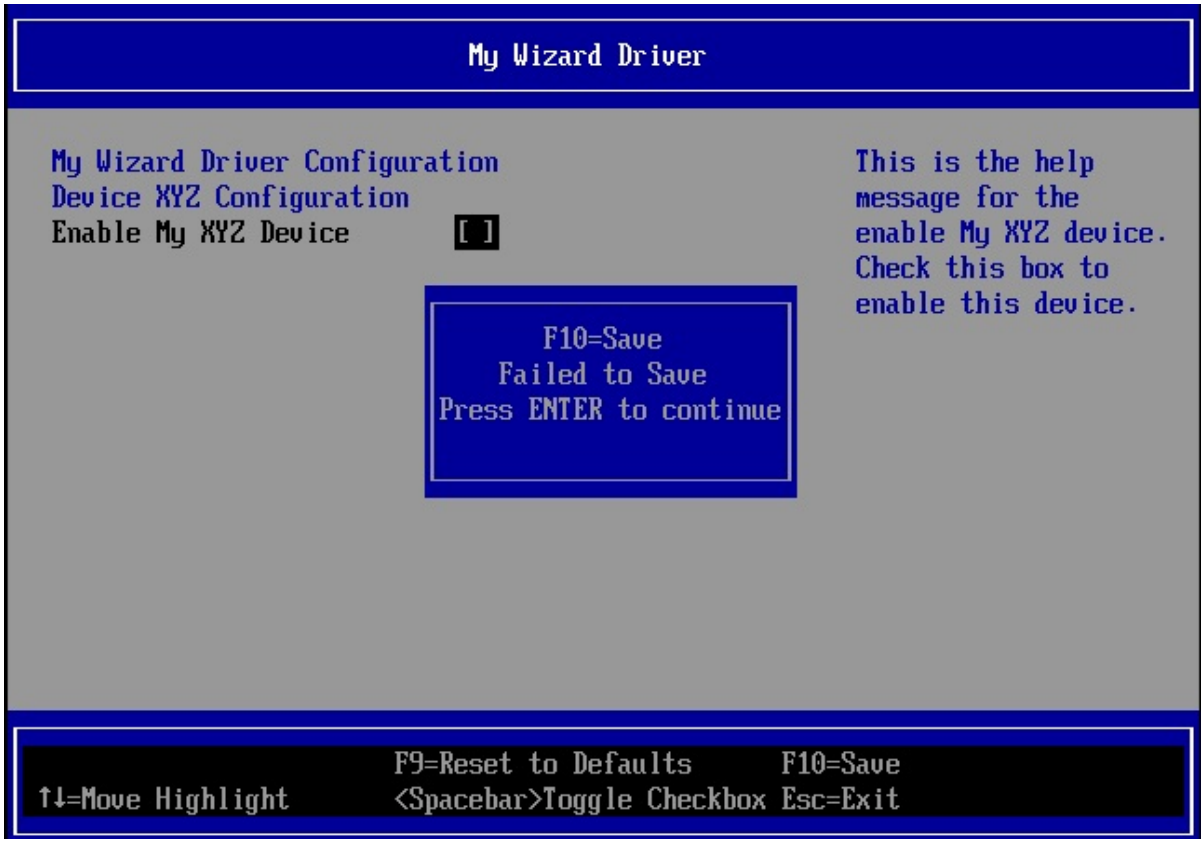
Press "Enter"



***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.

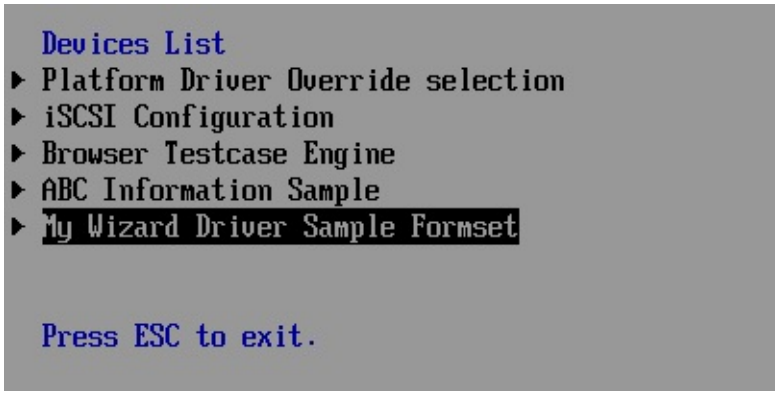
10. **Press** the space bar to Enable and Disable the “Enable My XYZ Device”

11. **Press** F10 to attempt to save

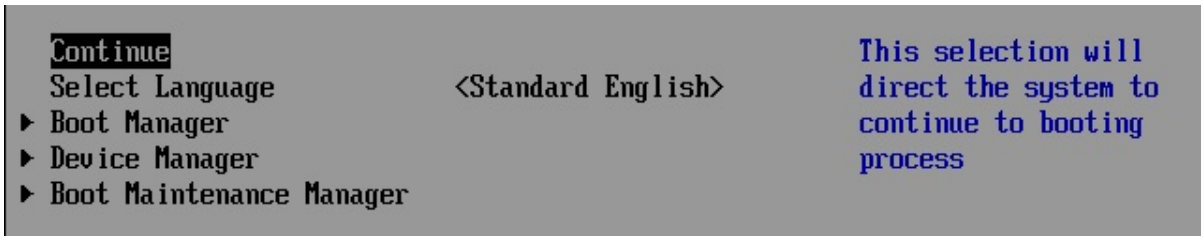


***Note*:** You're not able to save the data changes at this point.

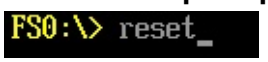
12. **Press** "Enter"**
13. **Press** "Escape", and then "Y" to exit*
14. **To Exit the "Device Manager" Page: Press "Escape"**



15. **Press** Up Arrow to "Continue"**



16. **At the Shell prompt** type **Reset**



17. **Exit** QEMU

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 ~/FW/LabSolutions/LessonE.1

End of Lab 1

Lab 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 ~src/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.

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

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

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

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

```
//Begin code
 //
  // Locate Hii Database protocol
  //
  Status = gBS->LocateProtocol (&gEfiHiiDatabaseProtocolGuid, NULL, (VOID **) &HiiData
base);
  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 **) &FormBr
owser2);
  if (EFI_ERROR (Status)) {
    return Status;
  }
  PrivateData->FormBrowser2 = FormBrowser2;

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

//End code
```

```
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, NUL
249   if (EFI_ERROR (Status)) {
250     return Status;
251   }
252   PrivateData->FormBrowser2 = FormBrowser2;
253
254   //
255   // Locate ConfigRouting protocol
256   //
257   Status = gBS->LocateProtocol (&gEfiHiiConfigRoutingProtocolGuid
258   if (EFI_ERROR (Status)) {
259     return Status;
260   }
261   PrivateData->HiiConfigRouting = HiiConfigRouting;
262
263
264   //
265   // Publish sample Fromset and config access
266   //
267   Status = gBS->InstallMultipleProtocolInterfaces (
```

4). Since the Hii Database Protocol was located earlier in the code with the previous code insertion and is no longer necessary, **comment out** the old OpenProtocol code with the “ `//` ” (approx. lines 289-298, as shown below) and **add the comment** " `//` " Done above Make sure not to comment out the second “ `if (!EFI_ERROR (Status)) {` ”

```
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)) {
```

Note: The earlier `LocateProtocol` code already found the pointer to the Hii Database protocol and stored it to the local pointer variable `HiiDatabase`.
When we added the driver-consumed protocols, we searched via `LocateProtocol` for the Hii Database pointer function. Since we did it above we're now commenting out this code.

5). **Comment out** the matching " `}` " with " `//` " to the if statement (as shown below at approx. line 310):

```
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
```

6). **Save** MyWizardDriver.c

7). **Open** ~src/edk2/MyWizardDriver/HiiConfigAccess.c.
The Driver Wizard only made dummy functions for the extract, route and callback functions. In order to save the Data passed into the forms from the Hii Browser engine, you will need to port these functions to be functional.

8). **Add** the following extern statements for the form GUID and the NVRam variable (as shown below) these are global to the driver module only hence the beginning lower case "m" is the standard for a global for a module :

```
extern EFI_GUID   mMyWizardDriverFormSetGuid;
extern CHAR16     mIfrVariableName[];
```

```
12  #include "MyWizardDriver.h"
13
14  extern EFI_GUID   mMyWizardDriverFormSetGuid;
15  extern CHAR16     mIfrVariableName[];
16
17
18  ///
19  /// HII Config Access Protocol instance
```

9). **Locate** `MyWizardDriverHiiConfigAccessExtractConfig` and **replace** line 108, “ `return EFI_NOT_FOUND` ”, with the following code spread over **Next** pages:

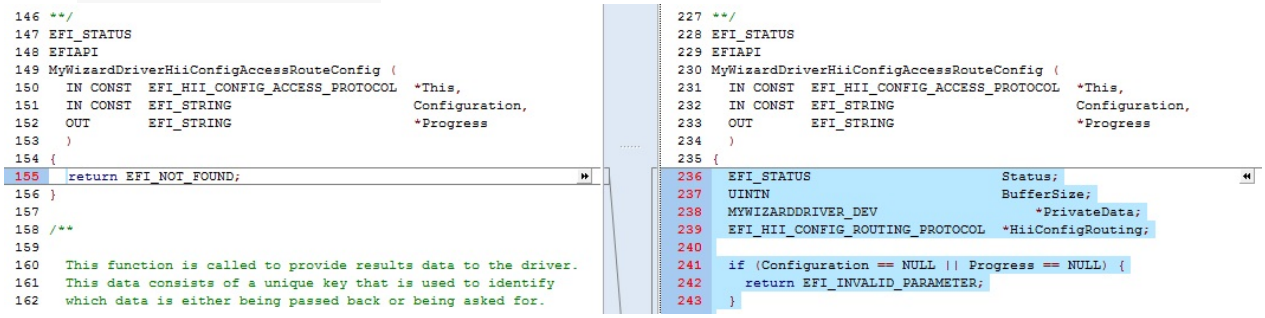
```
95  EFI_STATUS
96  EFIAPI
97  MyWizardDriverHiiConfigAccessExtractConfig (
98    IN CONST  EFI_HII_CONFIG_ACCESS_PROTOCOL  *This,
99    IN CONST  EFI_STRING                      Request,
100   OUT       EFI_STRING                      *Progress,
101   OUT       EFI_STRING                      *Results
102   )
103 {
104   return EFI_NOT_FOUND;
105 }
106
107 /**
```

```
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;
```

```
//Begin code
 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
             );
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;
// End code
```

10). Now **locate** `MyWizardDriverHiiConfigAccessRouteConfig` and **replace** line at approx. 228, with “ `return EFI_NOT_FOUND` ”, with the following code:

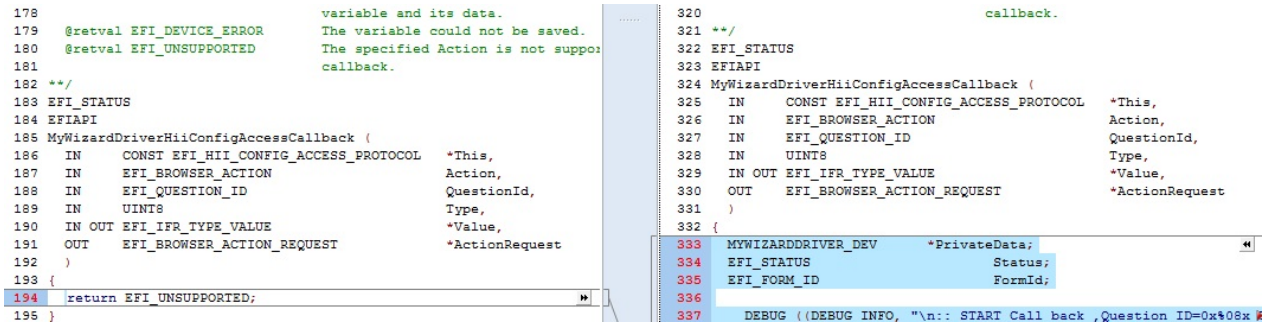



```
//Begin code
EFI_STATUS                        Status;
  UINTN                             BufferSize;
  MYWIZARDDRIVER_DEV                *PrivateData;
  EFI_HII_CONFIG_ROUTING_PROTOCOL   *HiiConfigRouting;
```

```
  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;
//end code
```

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




```
//Begin code
  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=0
x%04x", QuestionId, Type, Action));



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


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

  return Status;
//end code
```

12). **Save** HiiConfigAccess.c

Build and test MyWizardDriver

1. At the Terminal Command Prompt (**Cntl-Alt-T**)

```
bash$ cd ~/src/edk2
bash$ build
```

2. **Copy** MyWizardDriver.efi to hda-contents

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

3. **Invoke** Qemu

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

4. **At the UEFI Shell prompt,type** fs0:
5. **Type** Load MyWizardDriver.efi
6. **Press** “Enter”

```
FS0:\> load MyWizardDriver.efi
Image 'FS0:\MyWizardDriver.efi' loaded at 5EB9000 - Success
FS0:\> exit_
```

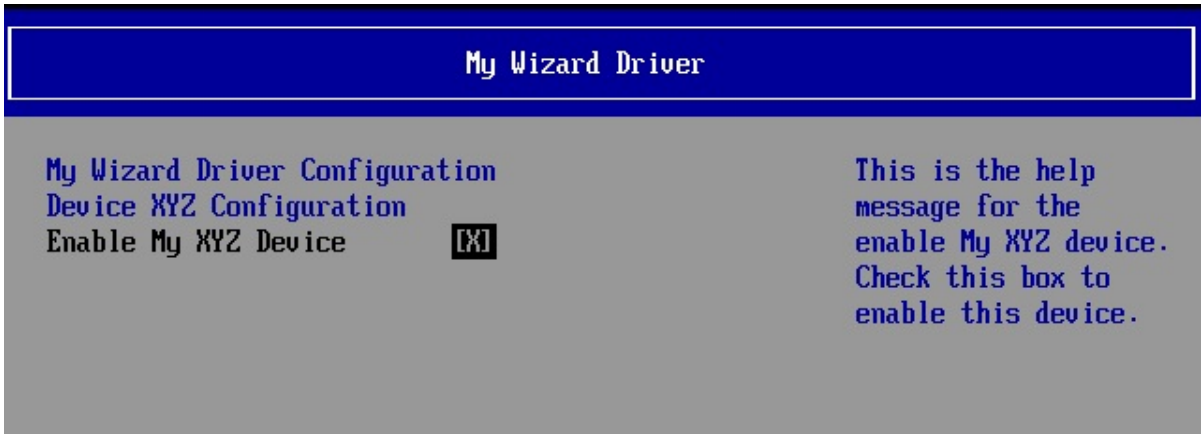
7. **Type** exit
8. Now at the setup front page menu, **select** “Device Manager”

```
  Continue                                    This selection will
  Select Language          <English>          take you to the
▶ Boot Manager                               Device Manager
▶ Device Manager
▶ Boot Maintenance Manager
```

9. **Press** “Enter” to enter **“Device Manager”**
10. **Inside the Device Manager menu** press **the down to “My Wizard Driver Sample Formset”** Press** "Enter"

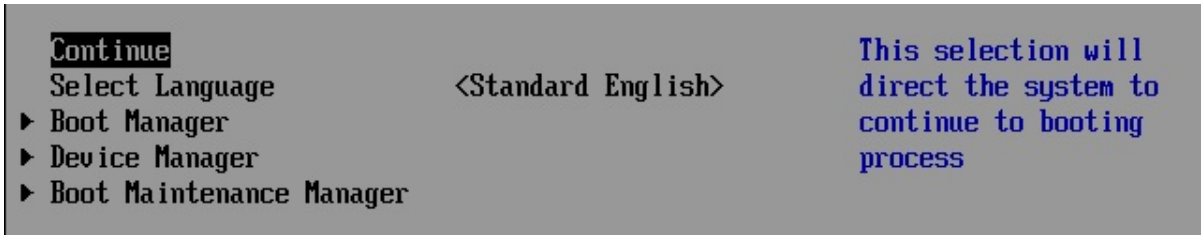
```
                        Device Manager

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

  Press ESC to exit
```

Press "Enter"



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.

11. Test by **Press** the space bar to Enable and Disable the “Enable My XYZ Device” to change its value from**:** `[X]` to `[ ]`
    Note: Notice the “ `Configuration changed` ” message at the menu bottom.
12. **Press** “F10”
13. **Press** “Escape” to exit
14. **Press** “Escape” to exit the “Device Manager” Page
15. **Press** Up Arrow to “Continue”



16. **Press** “Enter”
17. At the Shell Prompt type `Shell> dmpstore -all`
    Notice that enable is selected and saved in NVRam as the value of 0x00:


```
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;
```

Because our data structure in `MyWizardDriverNVDataStruc.h` is stored in NVRAM with the variable name `MWD_IfrNVData` of type `MYWIZARDDRIVER_CONFIGURATION` , we can see the

changes from our menu accessing through our HII forms.

Notice that the enable/disable byte is the last byte in data structure `MWD_IfrNVData.MyWizardDriverChooseToEnable` where `00 == disable` and `01 == enable`.

18. **Type** Reset

```
FS0:\> reset_
```

1. **Exit** Qemu

For any build issues copy the solution files from ~/Fw/LabSolutions/LessonE.2

NOTE: Del Directory ~/src/edk2/Build/OvmfX64/DEBUG_GCC5/X64/MyWizardDriver before the Build command to build the MyWizardDriver Clean

End of Lab 2

Lab 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".

Lab 3*a*. Add HII Library Calls to Your Driver

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

1. **Update** the MyWizardDriver.inf file
2. **Add** the following package (as shown below):
   The HII Library in the MdeModulePkg has many functions to help with Communication to/from the Hii Database and Hii forms. One function call HiiSetToDefaults will compare the default settings from the .VFR file and update the driver's configuration buffer according to the settings in the .VFR file.

```
MdeModulePkg/MdeModulePkg.dec
```

```
22 [Packages]
23   MdePkg/MdePkg.dec
24   MdeModulePkg/MdeModulePkg.dec
```

Note: For other functions from the HII Library, open the .chm file "MdeModulePkg Document.chm" and search for `HiiLib.h` .

3). **Add** the following library class (as shown below):

```
HiiLib
```

```
39 [LibraryClasses]
40   UefiDriverEntryPoint
41   UefiBootServicesTableLib
42   MemoryAllocationLib
43   BaseMemoryLib
44   BaseLib
45   UefiLib
46   DevicePathLib
47   DebugLib
48   HiiLib
```

4). **Save** MyWizardDriver.inf

5). **Update** the MyWizardDriver.h file

6). **Add** the following code (as shown below):

```
#include <Library/HiiLib.h>
```

```
42  // Added for HII
43  #include <Protocol/HiiConfigRouting.h>
44  #include <Protocol/FormBrowser2.h>
45  #include <Protocol/HiiString.h>
46  #include <Library/DevicePathLib.h>
47  #include <Library/HiiLib.h>
```

7). **Save** MyWizardDriver.h

8). **Update** the MyWizardDriver.c file

9). **Add Locals:** first add 2 locals for your drivers configuration buffer and a boolean flag from the Hii Library calls
Add the following at Approx. Line 190.

```
MYWIZARDDRIVER_CONFIGURATION     *Configuration;
BOOLEAN                          ActionFlag;
```

```
180
181   // HII Locals
182   EFI_HII_PACKAGE_LIST_HEADER     *PackageListHeader;
183   EFI_HII_DATABASE_PROTOCOL       *HiiDatabase;
184   EFI_HII_HANDLE                  HiiHandle[2];
185   EFI_HII_STRING_PROTOCOL         *HiiString;
186   EFI_FORM_BROWSER2_PROTOCOL      *FormBrowser2;
187   EFI_HII_CONFIG_ROUTING_PROTOCOL *HiiConfigRouting;
188   EFI_STRING                       ConfigRequestHdr;
189   UINTN                            BufferSize;
190   MYWIZARDDRIVER_CONFIGURATION     *Configuration;
191   BOOLEAN                          ActionFlag;
192
193   Status = EFI_SUCCESS;
194
```

10). **Add** the following to the MyWizardDriverDriverEntryPoint entry point funtion to line 319, approximately after “ `BufferSize =` ” as shown below

```
// Begin code

  //
  // Initialize configuration data
  //
  Configuration = &PrivateData->Configuration;
  ZeroMem (Configuration, sizeof (MYWIZARDDRIVER_CONFIGURATION));

  //
  // Try to read NV config EFI variable first
  //
  ConfigRequestHdr = HiiConstructConfigHdr (&mMyWizardDriverFormSetGuid, mIfrVariableN
ame, mDriverHandle[0]);
  ASSERT (ConfigRequestHdr != NULL);
// End code
```

```
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, mIfrVa
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 (
```

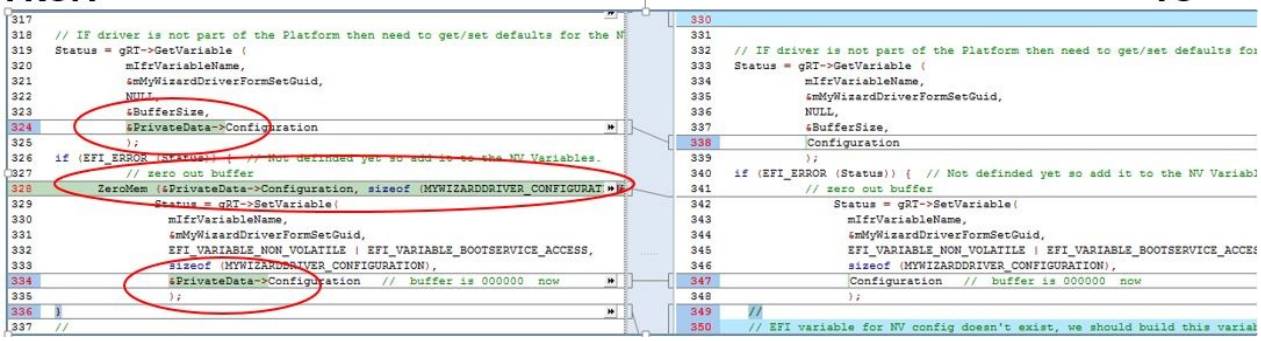
11). Modify the following lines:

`@~338` : remove: “ `&PrivateData->` ” from the “ `&PrivateData->Configuration` ”

`@~342` : remove line: `ZeroMem (&PrivateData->Configuration, sizeof (MYWIZARDDRIVER_CONFIGURATION));`

`@~347` : remove: “ `&PrivateData->` ” from the “ `&PrivateData->Configuration` ”

FROM **TO**



12). **Add** the following code to the `MyWizardDriverDriverEntryPoint` entry point code at approximately line 349 before

`// Install Driver Supported EFI Version Protocol onto ImageHandle` You’re deleting the “ `}` "

and replacing it with the following code (as shown below). With this replacement we are adding an “ `else` ” to the “ `if` ” statement:
Note the “ `}` ” on line 361 is still matching the initial if statement. Make sure you do not have a duplicate “ `}` ”

```
//Begin code
    //
    // 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
// End code
```

```
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_STANI
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 (
```

13). **Save** the MyWizardDriver.c file

Build and test MyWizardDriver

1. At the Terminal Command Prompt (**Cntl-Alt-T**)

```
bash$ cd ~/src/edk2
bash$ build
```

2. **Copy** MyWizardDriver.efi to hda-contents

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

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

3. **Invoke** Qemu

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

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

4. **At the UEFI Shell prompt,type** fs0:
5. **Type** Load MyWizardDriver.efi and then **Press** “Enter”

```
FS0:\> load MyWizardDriver.efi
Image 'FS0:\MyWizardDriver.efi' loaded at 5EB9000 - Success
FS0:\> exit_
```

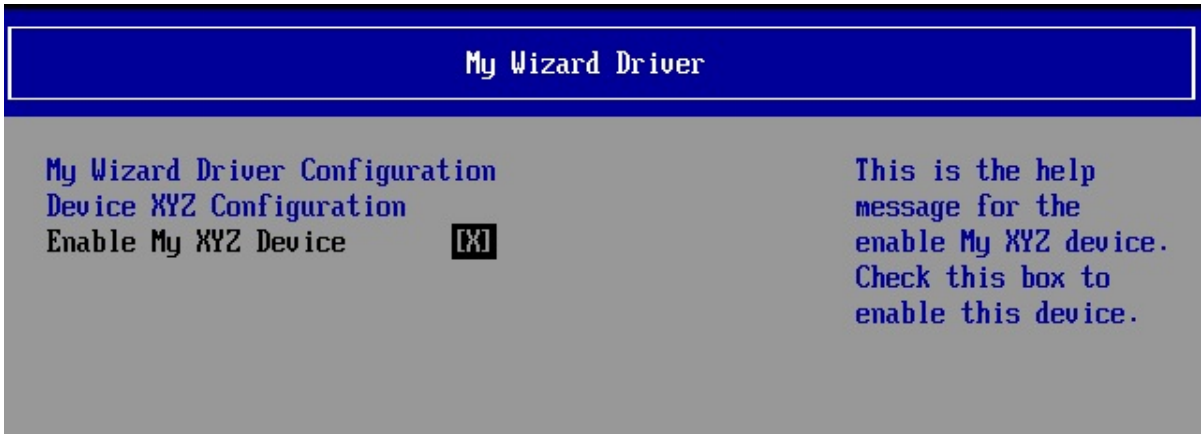
6. **Type** exit
7. Now at the setup front page menu, **select** “Device Manager”

```
  Continue                                    This selection will
  Select Language          <English>          take you to the
▶ Boot Manager                               Device Manager
▶ Device Manager
▶ Boot Maintenance Manager
```

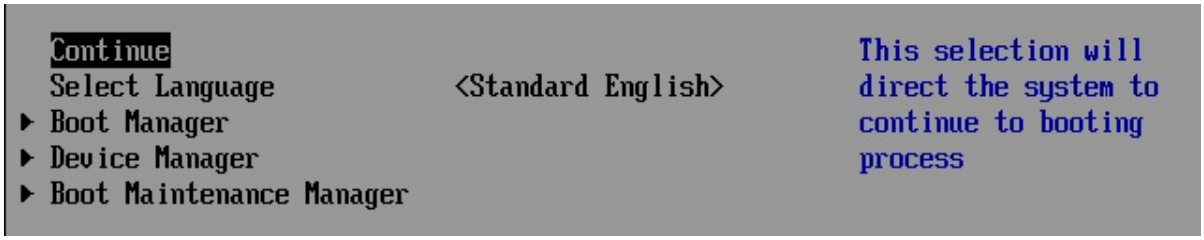
8. **Press** “Enter” to enter **“Device Manager”**
9. Inside the Device Manager menu **press** the down to **“My Wizard Driver Sample Formset”**

```
                        Device Manager

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

  Press ESC to exit
```

10. **Press** “Enter”



11. **Press** “Escape” to exit
12. **Press** “Escape” to Exit the “Device Manager” Page
13. **Press** Up Arrow to “Continue” and then **Press** “Enter”



14. **Type** " `Reset` "

15. **Exit** QEMU

For any build issues copy the solution files from ~/FW/LbSolutions/LessonE.3 NOTE: Del Directory ~/src/edk2/Build/OvmfX64/DEBUG_GCC5/X64/MyWizardDriver before the Build command to build the MyWizardDriver Clean

Lab 3*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.

1. **Open** to update: ~/src/edk2/OvmfPkg/OvmfPkgX64.Fdf
2. **Add** the following code (as shown below before “ `!if $(BUILD_NEW_SHELL) == TRUE` ”):

```
INF MyWizardDriver/MyWizardDriver.inf
```

3. **Save** ~/src/edk2/OvmfPkg/OvmfPkgX64.Fdf

Build and test MyWizardDriver

1. At the Terminal Command Prompt (**Cntl-Alt-T**)

```
bash$ cd ~/src/edk2
bash$ build
```

2. **Copy** the **Note:** `OVMF.fd` BIOS image created from the `build` to the run-ovmf directory naming it `bios.bin`

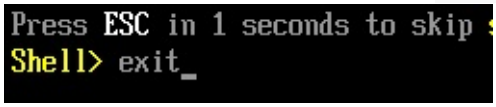
```
bash$ cp ~/src/edk2/Build/OvmfX64/DEBUG_GCC5/FV/OVMF.fd bios.bin
```

3. **Invoke** Qemu

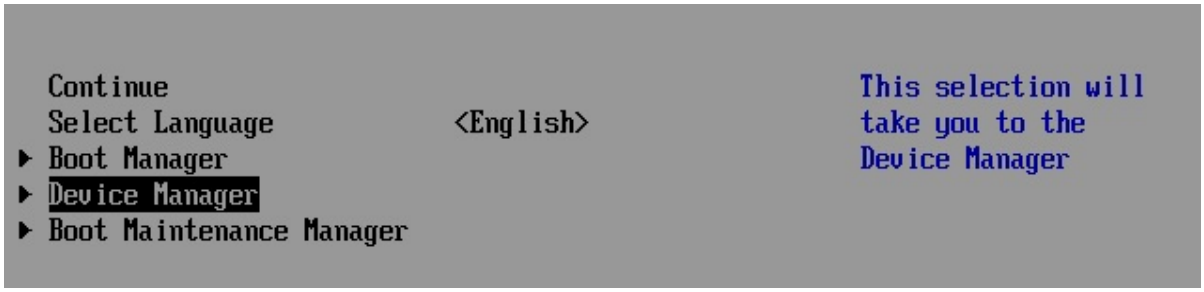
```
bash$ cd ~/run-ovmf
```

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

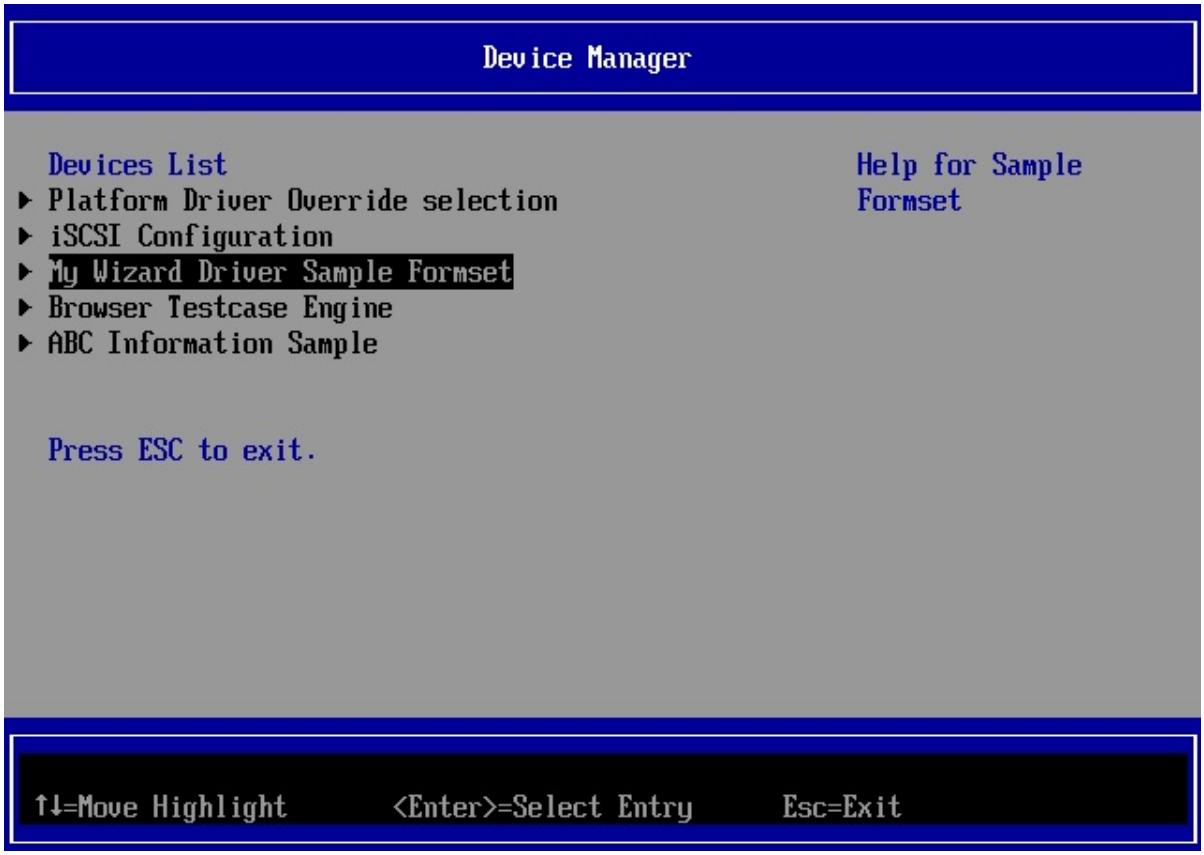
4. At the UEFI Shell prompt,**type** " `Exit` "



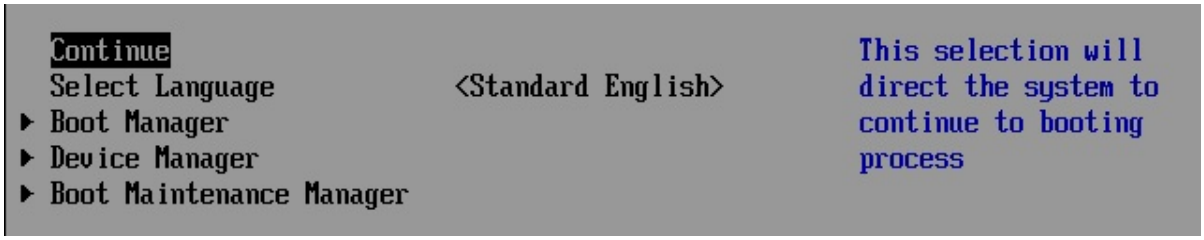
5. Now at the setup front page menu, **select** “Device Manager”



6. **Press** “Enter” to enter **“Device Manager”**
7. Inside the Device Manager menu **Notice** that the My Wizard Driver Sample Formset is added without having to issue the “Load” command from the shell prompt.



8. **Press** “Escape” to exit the “Device Manager” Page
9. **Press** Up Arrow to “Continue”



10. **Press** “Enter”
11. **Type** " `Reset` "

12. **Exit** QEMU

For any build issues copy the solution files from ~/FW/LbSolutions/LessonE.3

NOTE: Del Directory ~/src/edk2/Build/OvmfX64/DEBUG_GCC5/X64/MyWizardDriver before the Build command to build the MyWizardDriver Clean

End of Lab 3

Lab 4. Updating the Menu: Reset Button

In this lab, you'll learn how to add a resetbutton 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:

1). **Update** the MyWizardDriver.vfr file 2). **Add** the following code (as shown below after the "GUID" definition Apprx. Line 29): `MyStandardDefault` ,

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

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

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

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

```
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;
```

4). **Save** MyWizardDriver.vfr

5). **Update** the MyWizardDriver.uni file

6). **Add** the following strings at the end of the file to support the " `STR_` " referenced added in the .vfr file:

```
#string STR_STANDARD_DEFAULT_PROMPT    #language en "Standard Default"
#string STR_STANDARD_DEFAULT_PROMPT_RESET    #language en "Reset to Standard Default"
#string STR_STANDARD_DEFAULT_HELP      #language en "This will reset all the Questions
 to their standard default value"
```

7). **Save** MyWizardDriver.uni

Build and test MyWizardDriver

1. At the Terminal Command Prompt (**Cntl-Alt-T**)

   ```
   bash$ cd ~/src/edk2
   bash$ build
   ```

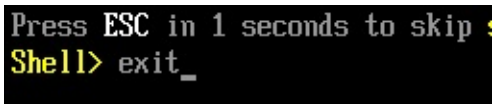
2. **Copy** the `OVMF.fd` BIOS image created from the `build` to the run-ovmf directory naming it `bios.bin`

   ```
   bash$ cp ~/src/edk2/Build/OvmfX64/DEBUG_GCC5/FV/OVMF.fd bios.bin
   ```

3. **Invoke** Qemu

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

4. At the UEFI Shell prompt,type **exit**



1. Now at the setup front page menu, **select** "Device Manager"
2. **Press** "Enter" to enter "Device Manager"
3. Inside the Device Manager menu **press** the down arrow to "My Wizard Driver Sample Formset" **Press** "Enter"
4. **Access** the My Wizard Driver menu and **notice** the item "Reset to Standard Default"

5. **Press Down Arrow to "**Reset to Standard Default"
6. **Press** "Enter" to select
7. **Notice** the "**Configuration changed**" message at the bottom of the menu
8. To Exit **Press** "Escape" then "Y"
9. To Exit the "Device Manager" Page: **Press** "Escape"
10. **Press** Up Arrow to "Continue"
    Observe: Notice that since this change requires a reset, the shell will exit out completely. 

11. **Press** "Enter" to reset
12. **Exit** QEMU

For any build issues copy the solution files from ~/FW/LabSolutions/LessonE.4

NOTE: Delete Directory ~/src/edk2/Build/OvmfX64/DEBUG_GCC5/X64/MyWizardDriver before the Build command to build the MyWizardDriver Clean

End of Lab 4

Lab 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.

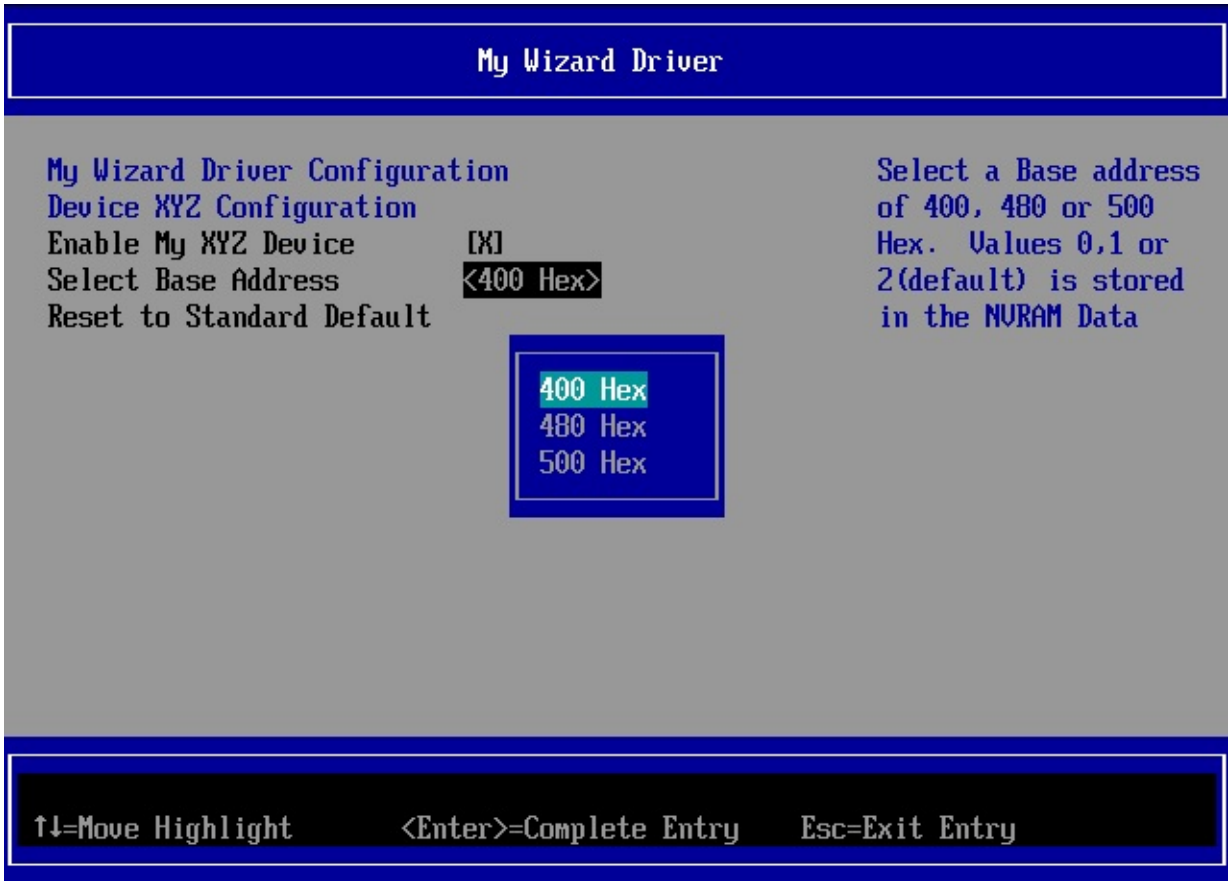


Figure 5 My Wizard Driver with a pop-up box

Background Information 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 5 above, there are three values:

Value	Display	String token
0	500 Hex	STR_ONE_OF_TEXT3
1	480 Hex	STR_ONE_OF_TEXT2
2	400 Hex	STR_ONE_OF_TEXT1

For this lab you will add code to give your driver menu a pop-up menu item by defining a "**oneof**" item. Also, if the device is "**disabled**", then use the VFR term "**grayoutif**" statement so that the pop-up menu is not accessible and cannot be changed. The browser engine will

use the configuration variable `MWD_IfrNVData.MyWizardDriverChooseToEnable` with a value of `0x0` to determine if the device is enabled or disabled

1. **Update** the MyWizardDriver.vfr file
2. **Add** the following code before the “ `resetbutton` ” statement (approximately line 53)

```
// Begin code
//
// 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;
// end code
```

```
51                  default  = 1,
52       endcheckbox;
53     //
54      // Define oneof (EFI_IFR_ONE_OF)
55      //
56     grayoutif  ideqval MWD_IfrNVData.MyWizardDriverChooseToEnable == 0x0;
57
58      oneof name = MyOneOf2,                                  // Define referenc
59        varid   = MWD_IfrNVData.MyWizardDriverBaseAddress,
60        // Use "DataStructure.Member" to reference Buffer Storage
61        prompt  = STRING_TOKEN(STR_ONE_OF_PROMPT),
62        help    = STRING_TOKEN(STR_ONE_OF_HELP),
63        //
64        // Define an option (EFI_IFR_ONE_OF_OPTION)
65        //
66        option text = STRING_TOKEN(STR_ONE_OF_TEXT3), value = 0x0, flags = 0;
67        option text = STRING_TOKEN(STR_ONE_OF_TEXT2), value = 0x1, flags = 0;
68        //
69        // DEFAULT indicate this option will be marked with
70        //  EFI_IFR_OPTION_DEFAULT
71        //
72        option text = STRING_TOKEN(STR_ONE_OF_TEXT1), value = 0x2,
73            flags = DEFAULT;
74      endoneof;
75     endif;
76
77
78      resetbutton
79        defaultstore = MyStandardDefault,
```

3. **Save** the MyWizardDriver.vfr file
4. **Update** the MyWizardDriver.uni file
5. **Add** the following code to the end of the file (as shown below):

```
// begin code
#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"

// end code
```

```
33 #string STR_STANDARD_DEFAULT_HELP      #language en "This will reset all the Questions to their standard default value"
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 Hex.  Values 0,1 or 2(default) is stored in the NVRAM Data"
38
39 #string STR_ONE_OF_TEXT1               #language en "400 Hex"
40
41 #string STR_ONE_OF_TEXT2               #language en "480 Hex"
42
43 #string STR_ONE_OF_TEXT3               #language en "500 Hex"
```

6). **Save** MyWizardDriver.uni

Build and test MyWizardDriver

1. At the Terminal Command Prompt (**Cntl-Alt-T**)

```
bash$ cd ~/src/edk2
bash$ build
```

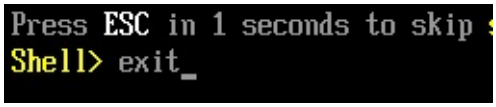
2. **Copy** the `OVMF.fd` BIOS image created from the `build` to the run-ovmf directory naming it `bios.bin`

```
bash$ cp ~/src/edk2/Build/OvmfX64/DEBUG_GCC5/FV/OVMF.fd bios.bin
```

3. **Invoke** Qemu

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

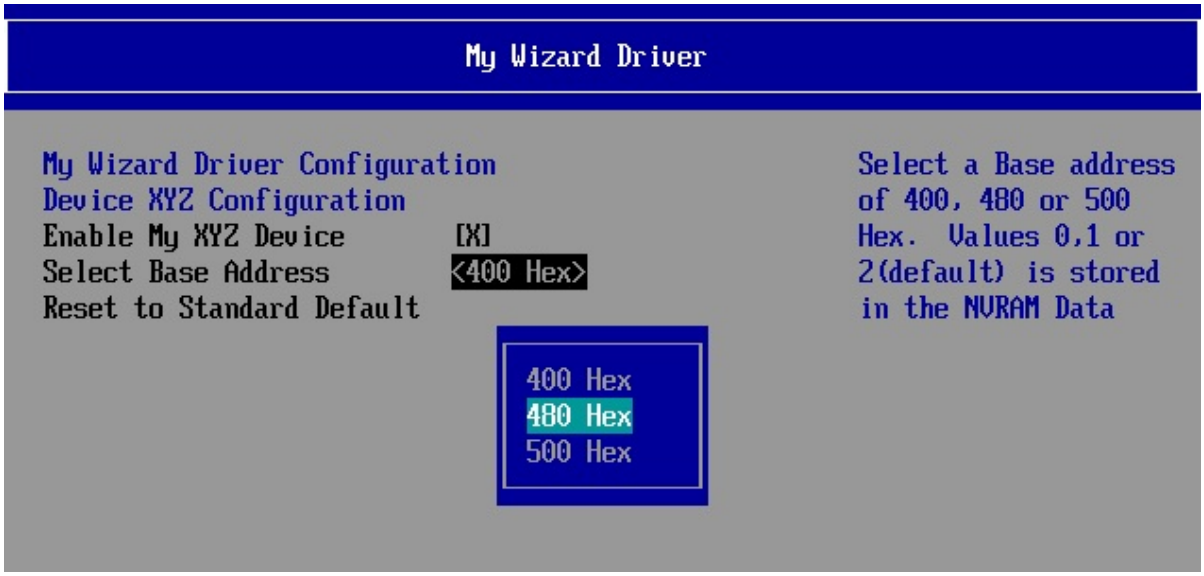
4. At the UEFI Shell prompt,type **exit**



5. Now at the setup front page menu, **select** “Device Manager”
6. Inside the Device Manager menu **press** the down arrow to “My Wizard Driver Sample Formset” **Press** "Enter"
7. **Down Arrow** to “Select Base Address”

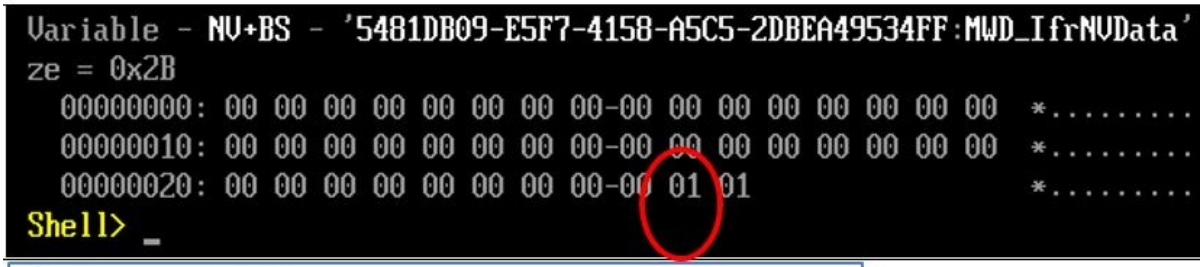
Notice the Pop up menu

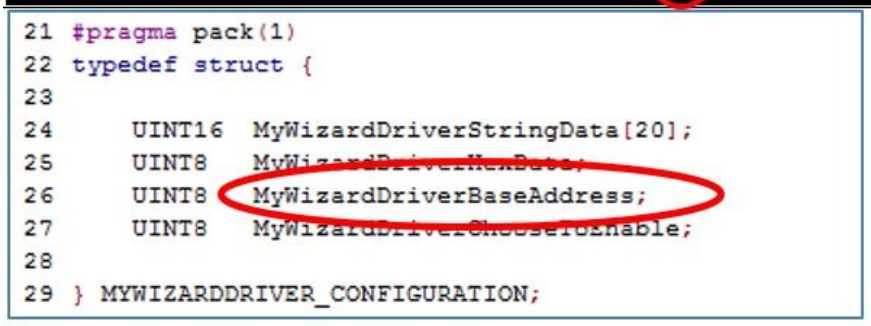
8. **Select** “480 Hex” then **press** "Enter"
Observe: the “**Configuration changed**” message at the bottom



9. **Test** the " `grayoutif` " by selecting "Enable My XYZ Device" then **press** the "Space" bar to toggle off or "Disabled".
   Notice the "Select Base Address" is now grayed out and not Selectable.

10. **Press** "Space" again to Enable
11. To Exit **Press** "Escape" then "Y" or "F10" then "Escape"
12. To Exit the "Device Manager" Page: **Press** "Escape"
13. **Press** Up Arrow to "Continue"
14. At the Shell Prompt **type**: " `dmpstore -all` "





```
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;
```

15. Observe file MyWizardDriverNVDataStruc.h and the NVRAM `MWD_IfrNVData` variable. By updating MyWizardDriverNVDataStruc.h, our data structure stored in NVRAM is named `MWD_IfrNVData` of type `MYWIZARDDRIVER_CONFIGURATION`.
    Notice that the base address byte is the next to the last byte in the data structure `MWD_IfrNVData.MyWizardDriverBaseAddress` where `02 == 400H`, `01 == 480H`, and `00 == 500H`
    Notice the NVRAM Variable with the value of `480H` will have a true value of `01`.
16. **Type** “reset” at the Shell prompt
    

17. **Exit** QEMU

For any build issues copy the solution files from ~/Fw/LabSolutions/LessonE.5

NOTE: Delete Directory ~/src/edk2/Build/OvmfX64/DEBUG_GCC5/X64/MyWizardDriver before the Build command to build the MyWizardDriver Clean

End of Lab 5

Lab 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.

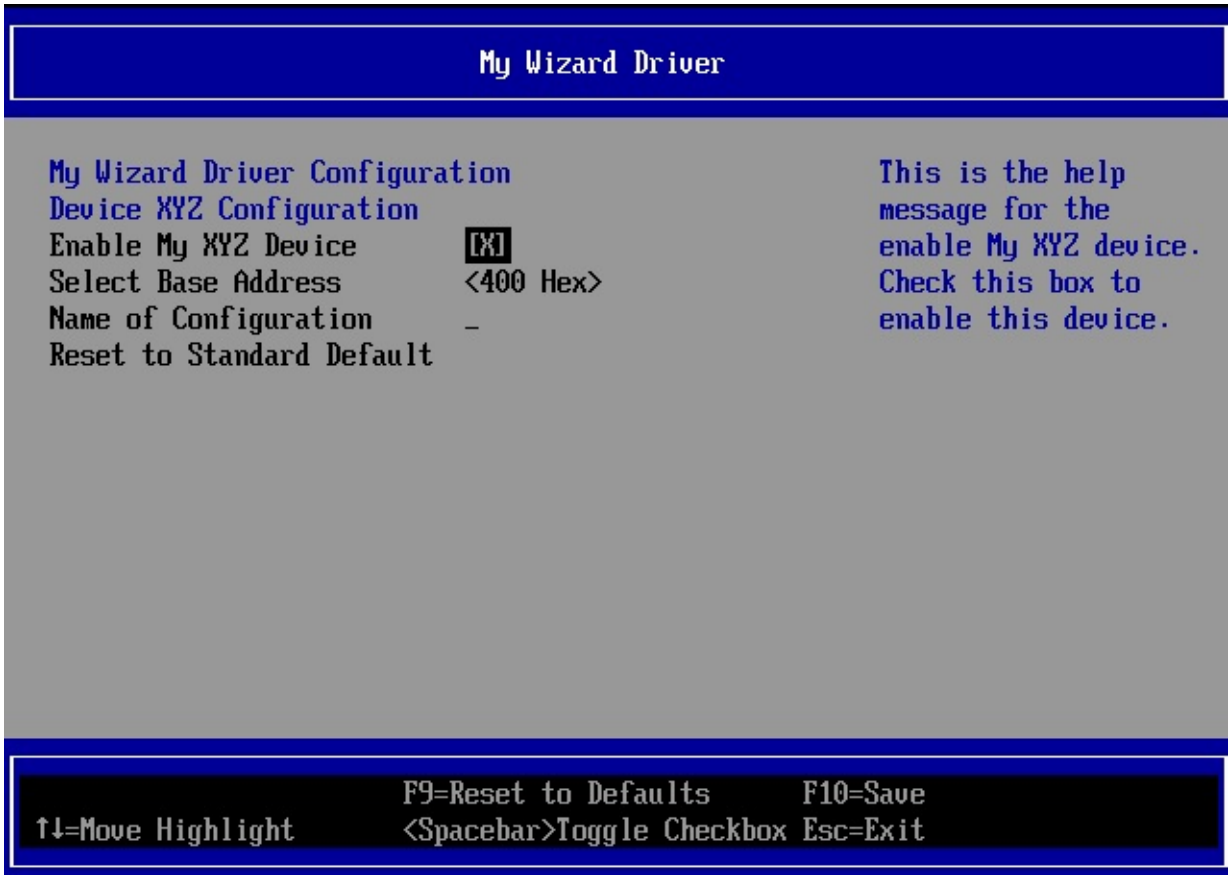


Figure 6 : Menu with a string item

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):

```
//
// Define a string (EFI_IFR_STRING) to name the configuration in the
// NVRAM variable
//
grayoutif  ideqval MWD_IfrNVData.MyWizardDriverChooseToEnable == 0x0;
string    varid    = MWD_IfrNVData.MyWizardDriverStringData,
          prompt   = STRING_TOKEN(STR_MY_STRING_PROMPT),
          help     = STRING_TOKEN(STR_MY_STRING_HELP),
          minsize  = 3,
          maxsize  = 20,
endstring;
endif;
```

```
74      endoneof;
75     endif;
76
77
78      //
79      // Define a string (EFI_IFR_STRING)to name the configuration in the
80      // NVRAM variable
81      //
82
83      //
84    grayoutif  ideqval MWD_IfrNVData.MyWizardDriverChooseToEnable == 0x0;
85
86      string    varid    = MWD_IfrNVData.MyWizardDriverStringData,
87                prompt   = STRING_TOKEN(STR_MY_STRING_PROMPT),
88                help     = STRING_TOKEN(STR_MY_STRING_HELP),
89                minsize  = 3,
90                maxsize  = 20,
91
92      endstring;
93    endif;
94
95      resetbutton
96        defaultstore = MyStandardDefault,
97        prompt   = STRING_TOKEN(STR_STANDARD_DEFAULT_PROMPT_RESET),
98        help     = STRING_TOKEN(STR_STANDARD_DEFAULT_HELP),
99      endresetbutton;
100
```

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

```
#string STR_MY_STRING_PROMPT           #language en "Name of Configuration"
#string STR_MY_STRING_HELP             #language en "Enter a name for this configurati
on. This string can is 3 to 20 characters"
```

```
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
```

6). **Save** MyWizardDriver.uni

Build and test MyWizardDriver

1. At the Terminal Command Prompt (**Cntl-Alt-T**)

```
bash$ cd ~/src/edk2
bash$ build
```

2. **Copy** the `OVMF.fd` BIOS image created from the `build` to the run-ovmf directory naming it `bios.bin`

```
bash$ cp ~/src/edk2/Build/OvmfX64/DEBUG_GCC5/FV/OVMF.fd bios.bin
```

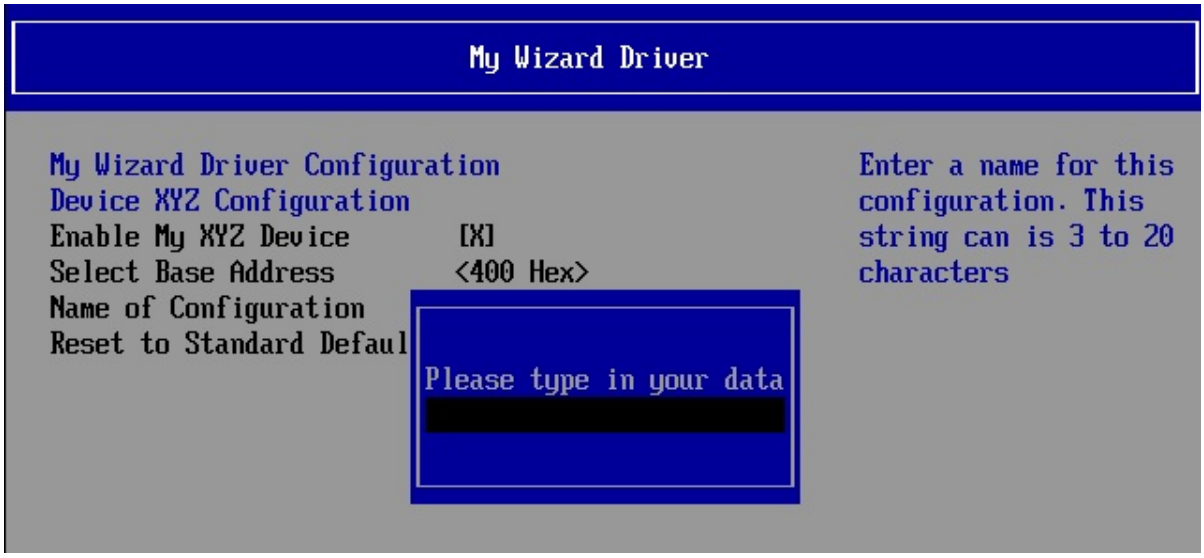
3. **Invoke** Qemu

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

4. At the UEFI Shell prompt,type **exit**

```
Press ESC in 1 seconds to skip
Shell> exit_
```

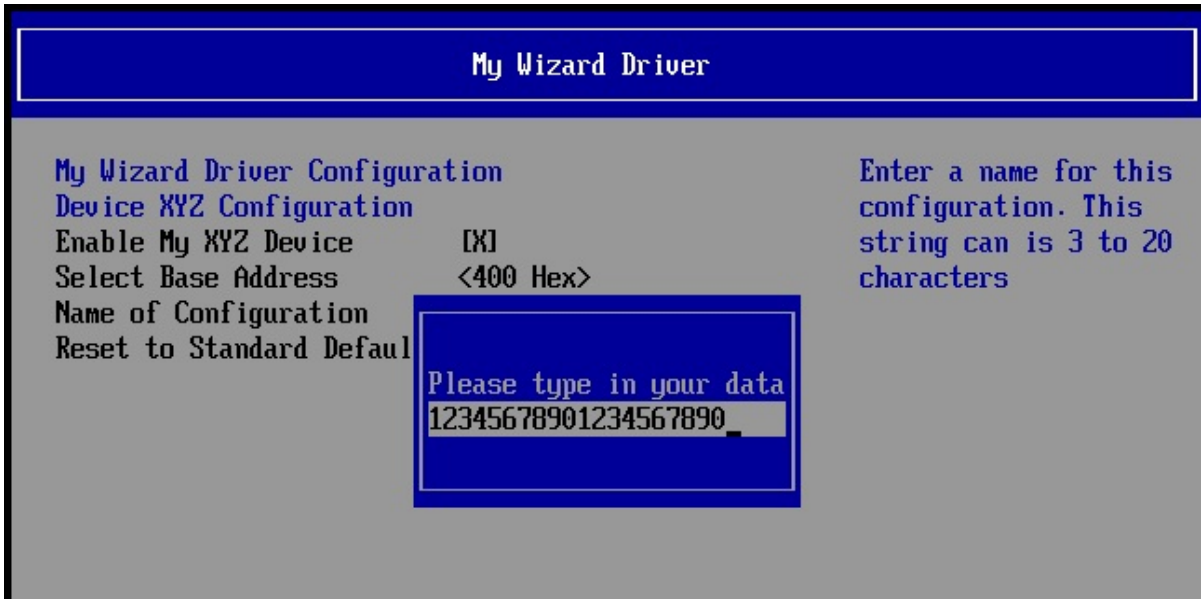
5. Now at the setup front page menu, **select** “Device Manager”
6. Inside the Device Manager menu **press** the down arrow to “My Wizard Driver Sample Formset” **Press** "Enter"
7. **Select** “Name of Configuration” then **Press** “Enter”
   Notice the string text pop up menu gets displayed.



8. **Test** the “ `minsize` ” by only **typing** your choice of a two character string. Then **Press** “Enter”

   Notice the an error message validating that you need to enter at least three or more characters.

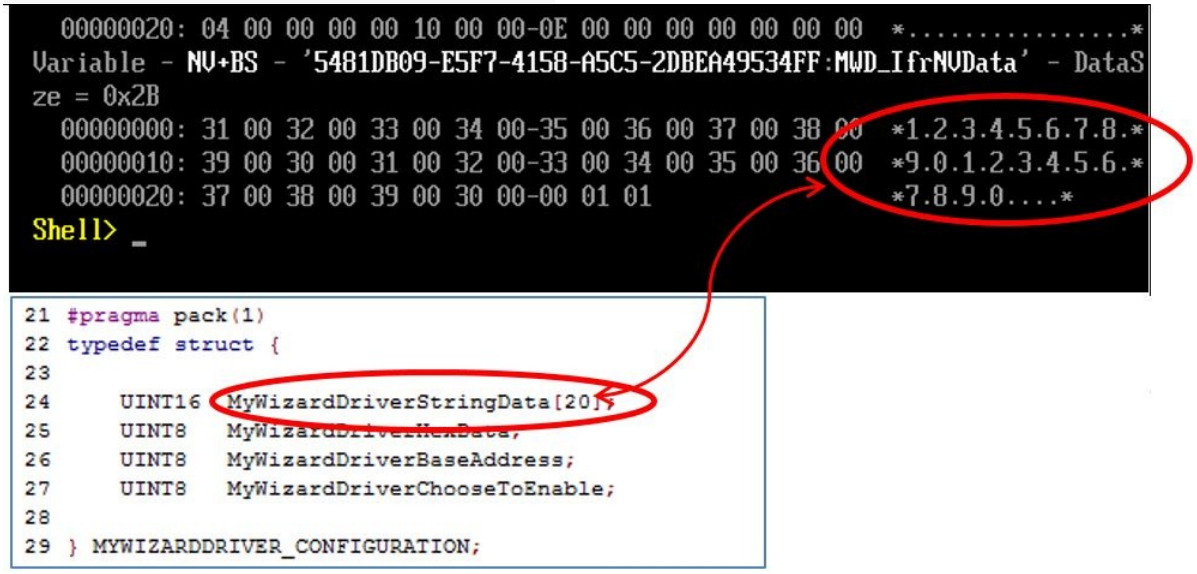
9. **Press** “Enter” to clear the message
10. **Press** “Enter” again to re-enter pop up menu
11. **Test** by **typing** more than “ `maxsize` ” of 20 characters**** Then** Press **“Enter” .** Notice** that the Browser only allows the maximum number of 20 characters to be entered with a forced stop. There is no error message but no more characters are allowed to be typed into the pop up menu.



12. **Press** “Enter”
    Notice that the “**Configuration changed**” message is displayed

13. **Test** the `grayoutif` by selecting **“Enable My XYZ Device”**
14. **Press** the “Spacebar” to toggle off/disable. Then **Press** the “Spacebar” again to Enable.
    Notice that the “Select Base Address” and “Name of Configruation” fields are now grayed out and not selectable

15. **Press** “F10” to save.
16. **Press** “Escape” to exit
17. **Press** “Escape” to exit the “Device Manager”
18. **Select** “Continue” and then **Press** “Enter”
19. At the Shell Prompt, **type** `dmpstore -all`
    Notice the unicode string “ `12345678901234567890` ” (or the 20 character value you typed) 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` .



20. **Type** “reset” and then “Enter" at the Shell prompt

21. **Exit** QEMU

For any build issues copy the solution files from ~/Fw/LabSolutions/LessonE.6

NOTE: Delete Directory ~/src/edk2/Build/OvmfX64/DEBUG_GCC5/X64/MyWizardDriver before the Build command to build the MyWizardDriver Clean

End of Lab 6

Lab 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

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

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

```
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 B
ase address in Hex. The valid range in this case is from 0 to 250."
```

6). **Save** MyWizardDriver.uni

Build and test MyWizardDriver

1. At the Terminal Command Prompt (**Cntl-Alt-T**)

```
bash$ cd ~/src/edk2
bash$ build
```

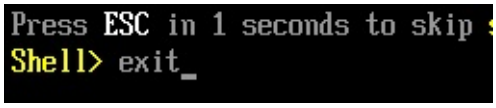
2. **Copy** the `OVMF.fd` BIOS image created from the `build` to the run-ovmf directory naming it `bios.bin`

   ```
   bash$ cp ~/src/edk2/Build/OvmfX64/DEBUG_GCC5/FV/OVMF.fd bios.bin
   ```

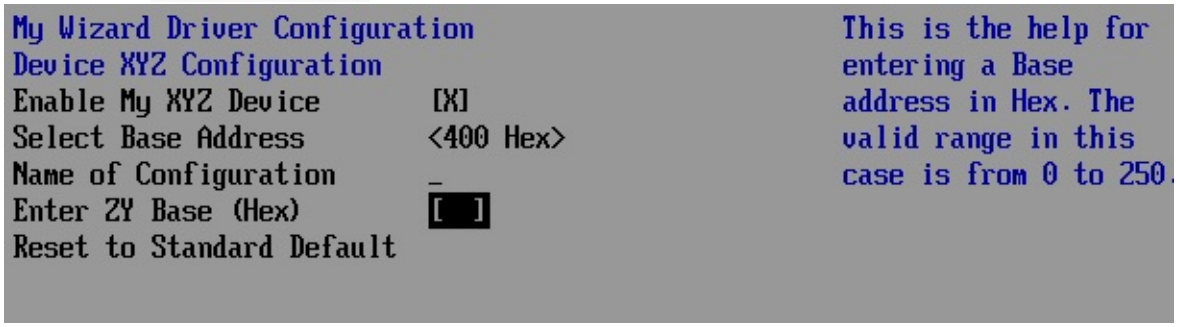
3. **Invoke** Qemu

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

4. At the UEFI Shell prompt,type **exit**

5. Now at the setup front page menu, **select** “Device Manager”
6. Inside the Device Manager menu **press** the down arrow to “My Wizard Driver Sample Formset” **Press** "Enter"
   Notice the value for “Enter ZY Base(Hex)” is `022` . Hex is the default because of the VFR field “ `default = 0x22` ”

7. **Select** “Enter ZY Base(Hex)” and then **Press** “Enter”
8. **Test** by typing a “ `M` ” character
   Notice that only Numeric characters are allowed and also only values `00` to `0FA` Hex. When values outside the range or non-numeric characters are entered the red **“!!”** sting is displayed at the bottom of the menu.
   The string **“!!”** is part of the Browser engine :

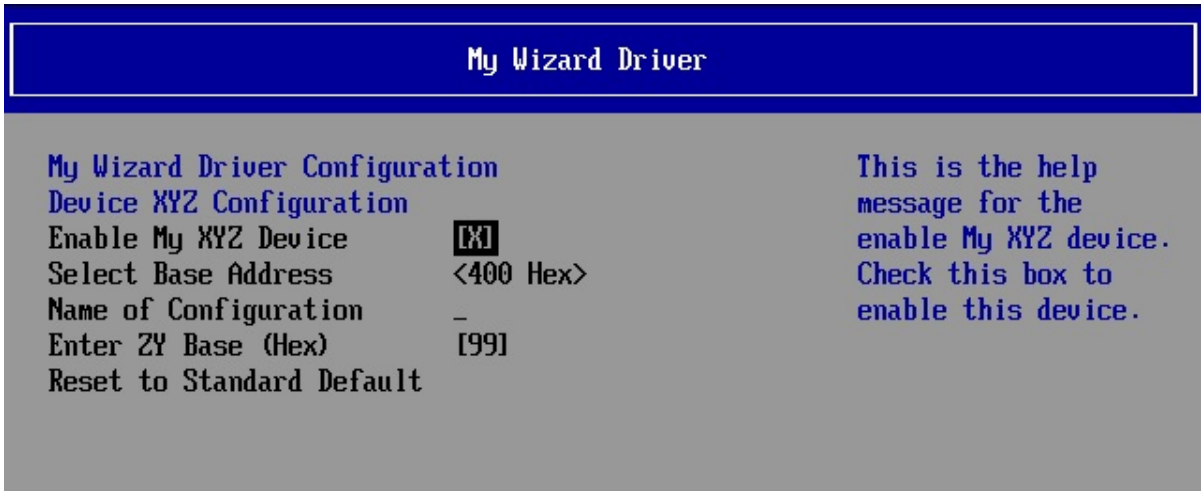
`MdeModulePkg\Universal\SetupBrowserDxe\SetupBrowserStr.uni`

`#string INPUT_ERROR_MESSAGE #language en-US "!!"`

9. **Press** “Enter” again

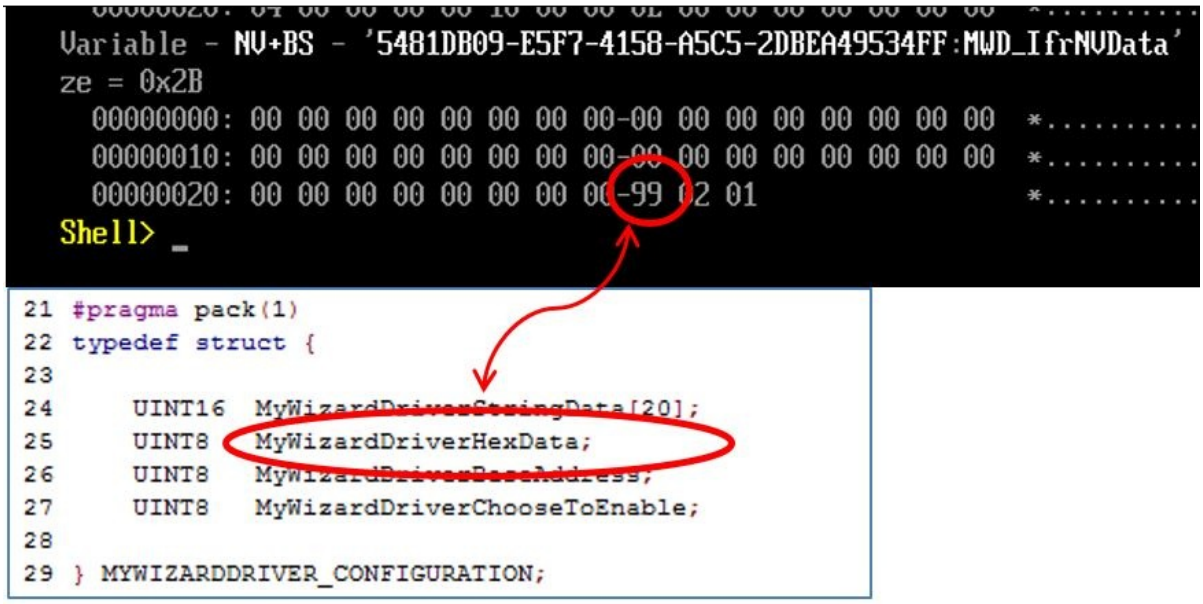
10. **Test** by typing a value of ‘ `99` ’ Hex and then **Press** “Enter”
    Notice that the “**Configuration changed**” message is displayed.
11. **Test** the “ `surpressif` ” by pressing the “spacebar” to “Enable My XYZ Device” then press the “Space” bar to toggle off or “Disabled”.
    Notice the “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.

12. **Press** “Space bar” again to “Enable My XYZ Device” and the“**Enter ZY Base(Hex)**” is displayed again



13. **Press** “F10” to save, then “Escape” to exit
14. **Press** “Escape” to exit the “Device Manager”
15. **Select “**Continue**”** and then **Press** "Enter”
16. At the Shell Prompt, **type** `dmpstore -all`

    Notice by `modifyingMyWizardDriverNVDataStruc.h` our data structure stored in NVRAM is named `MWD_IfrNVData` of type `MYWIZARDDRIVER_CONFIGURATION`.



17. **Type “reset”** at the Shell prompt

18. **Exit** QEMU

For any build issues copy the solution files from ~/FW/LabSolutions/LessonE.7

NOTE: Delete Directory ~/src/edk2/Build/OvmfX64/DEBUG_GCC5/X64/MyWizardDriver before the Build command to build the MyWizardDriver Clean

End of Lab 7

Lab 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%04
x", QuestionId, Type, Action));
```

Lab 8*a*. Add the Case statements to the Call back routine

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

```
//Begin Code
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;
//End Code
```

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

3. **Save** HiiConfigAccess.c

Build and test MyWizardDriver

1. At the Terminal Command Prompt (**Cntl-Alt-T**)

```
bash$ cd ~/src/edk2
bash$ build
```

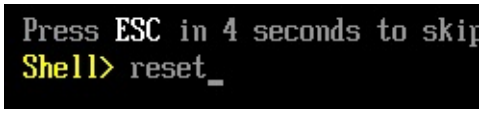
2. **Copy** the `OVMF.fd` BIOS image created from the `build` to the run-ovmf directory naming it `bios.bin`

   ```
   bash$ cp ~/src/edk2/Build/OvmfX64/DEBUG_GCC5/FV/OVMF.fd bios.bin
   ```

3. **Invoke** Qemu

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

4. At the UEFI Shell prompt,type **exit**
5. Now at the setup front page menu, **select** “Device Manager”
6. Inside the Device Manager menu **press** the down arrow to “My Wizard Driver Sample Formset” **Press** "Enter"
7. **Notice** that there are **No Debug messages for Call back** to the QEMU Console.
8. **Press** “Escape” and another "Escape" to exit the “Device Manager”
9. **Select** “Continue” and then **Press** "Enter”
10. **Type** “reset” at the Shell prompt

11. **Exit** QEMU

Lab 8*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,
   ```

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

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

```
questionid = 0x1111,
```

```
| INTERACTIVE ,
```

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

4. **Save** MyWizardDriver.vfr

Build and test MyWizardDriver

1. At the Terminal Command Prompt (**Cntl-Alt-T**)

```
bash$ cd ~/src/edk2
bash$ build
```

2. **Copy** the `OVMF.fd` BIOS image created from the `build` to the run-ovmf directory naming it `bios.bin`

```
bash$ cp ~/src/edk2/Build/OvmfX64/DEBUG_GCC5/FV/OVMF.fd bios.bin
```

3. **Invoke** Qemu

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

4. At the UEFI Shell prompt,type **exit**
5. Now at the setup front page menu, **select** “Device Manager”
6. Inside the Device Manager menu **press** the down arrow to “My Wizard Driver Sample Formset” **Press** "Enter"
7. Take a moment and **review** the QEMU Console window
8. In the emulation window, **click** on “Name of Configuration” and “Enter ZY Base(Hex)”
9. **Notice** the following in the QEMU Console window:
   Every time the browser does anything with the interactive labeled fields there is a call made to your driver’s call back function. We can determine which item by the `quetionid` and what action by the Action passed to your call back function. Your call

back function can then add code to special case when these transitions occur.

Entering Form

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

Changing a Value for Question ID 0x1111

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

Changing a Value for Question ID 0x1001

```
:: START Call back ,Question ID=0x00001001 Type=0x0007 Action=0x0003
:: START Call back ,Question ID=0x00001111 Type=0x0000 Action=0x0003
:: START Call back ,Question ID=0x00001001 Type=0x0007 Action=0x0002
:: START Call back ,Question ID=0x00001111 Type=0x0000 Action=0x0002
:: START Call back ,Question ID=0x00001111 Type=0x0000 Action=0x0000
:: START Call back ,Question ID=0x00001111 Type=0x0000 Action=0x0001
:: START Call back ,Question ID=0x00001001 Type=0x0007 Action=0x0002
:: START Call back ,Question ID=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
:: 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=0x0004
colInterface: 348C4D62-BFBD-4882-9ECE-C80BB1C4783B 0
```

10. **Press** “Escape” and another "Escape" to exit the “Device Manager”
11. **Select** “Continue” and then **Press** "Enter”
12. **Type** “reset” at the Shell prompt

```
FS0:\> reset_
```

13. **Exit** QEMU

For any build issues copy the solution files from ~/FW/LabSolutions/LessonE.8

NOTE: Delete Directory ~/src/edk2/Build/OvmfX64/DEBUG_GCC5/X64/MyWizardDriver before the Build command to build the MyWizardDriver Clean

End of Lab 8

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

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

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

1. **Update** the HiiConfigAccess.c file
2. **Comment out** the DEBUG statement with " `//` " in the `MyWizardDriverHiiConfigAccessCallback` call back function approx. line 330:

```
//
---
326   MYWIZARDDRIVER_DEV      *PrivateData;
327   EFI_STATUS                      Status;
328   EFI_FORM_ID                     FormId;
329
330 //    DEBUG ((DEBUG_INFO, "\n:: START Call back ,Question ID=0
331
332
333
334   if (((Value == NULL) && (Action != EFI_BROWSER_ACTION_FORM_O
```

3. **Add** a switch case statement of the question ID's to the "Action" switch case of `EFI_BROWSER_ACTION_CHANGING` in the call back function by **adding** a nested switch case code (as shown below at approx. line 372)

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

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

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

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

```
385
386   case EFI_BROWSER_ACTION_CHANGED:  // 1
387     {
388                                 switch (QuestionId) {
389                           case 0x1111:
390                                   DEBUG ((DEBUG_INFO, "\n:: START Call back- Changed
391                                 break;
392                           case 0x1001:
393                                   DEBUG ((DEBUG_INFO, "\n:: START Call back- Changed
394                                 break;
395                           default:
396                                 Status = EFI_UNSUPPORTED;
397                                 break;
398                   }
399         }
400     break;
401
402   default:
403     Status = EFI_UNSUPPORTED;
404     break;
405   }  // end switch case on Action
406
407   return Status;
408
```

1. **Save** MyWizardDriver.c

Build and test MyWizardDriver

1. At the Terminal Command Prompt (**Cntl-Alt-T**)

```
bash$ cd ~/src/edk2
bash$ build
```

2. **Copy** the `OVMF.fd` BIOS image created from the `build` to the run-ovmf directory naming it `bios.bin`

```
bash$ cp ~/src/edk2/Build/OvmfX64/DEBUG_GCC5/FV/OVMF.fd bios.bin
```

3. **Invoke** Qemu

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

4. At the UEFI Shell prompt,type **exit**
5. Now at the setup front page menu, **select** “Device Manager”
6. Inside the Device Manager menu **press** the down arrow to “My Wizard Driver Sample Formset” **Press** "Enter"
7. **Observe** the QEMU Console Window or debug output
   Test: in the your drivers menu by changing the “Name of Configuration” and the “Enter ZY Base(Hex)” fields while observing the QEMU Console Window or debug output.
8. **Observe** in the QEMU Console Window or debug output the changes that you made.

```
:: START Call back- Changing ,Question ID=0x00001001 Type=0x0007 Action=0x0000
```

Notice: when changing the “**Name of Configuration**” field

```
:: START Call back- Changing ,Question ID=0x00001001 Type=0x0007 Action=0x0000
:: START Call back- Changed ,Question ID=0x00001001 Type=0x0007 Action=0x0001
:: START Call back- Changing ,Question ID=0x00001111 Type=0x0000 Action=0x0000
```

Notice: when changing the “**Enter ZY Base(Hex)**” field

```
:: START Call back- Changing ,Question ID=0x00001001 Type=0x0007 Action=0x0000
:: START Call back- Changed ,Question ID=0x00001001 Type=0x0007 Action=0x0001
:: START Call back- Changing ,Question ID=0x00001111 Type=0x0000 Action=0x0000
:: START Call back- Changed ,Question ID=0x00001111 Type=0x0000 Action=0x0001
:: ROUTE CONFIG Saving the configuration to NVRAM
```

Notice: when Pressing “F10”

9. **Press** “Escape” and another "Escape" to exit the “Device Manager”
10. **Select** “Continue” and then **Press** "Enter”
11. **Type** “reset” at the Shell prompt

```
FS0:\> reset_
```

12. **Exit** QEMU

For any build issues copy the solution files from ~/FW/LabSolutions/LessonE.9

NOTE: Delete Directory ~/src/edk2/Build/OvmfX64/DEBUG_GCC5/X64/MyWizardDriver before the Build command to build the MyWizardDriver Clean

End of Lab 9

Lab 10. Adding an Additional Form Page

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

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

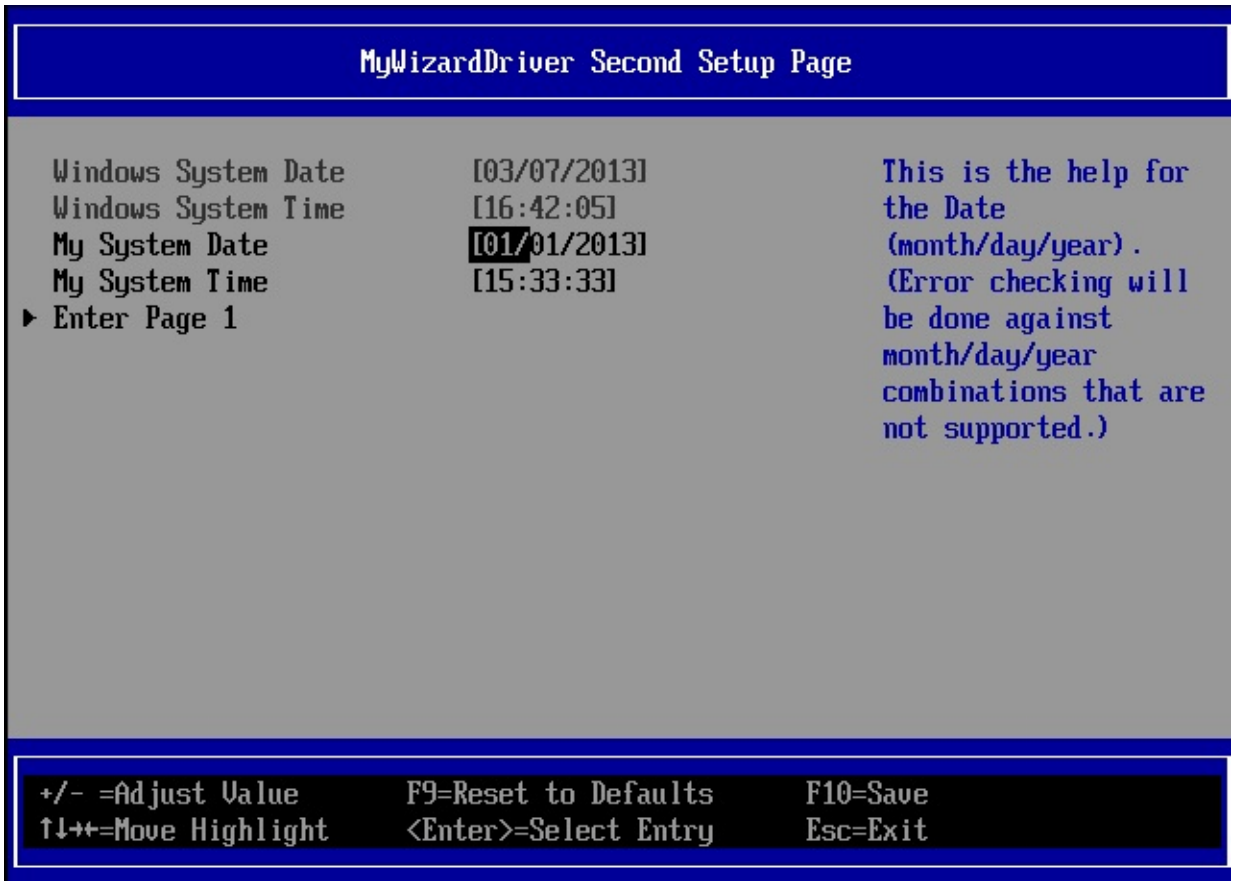


Figure 10: Second setup page

1. **Update** the MyWizardDriverNVDataStruc.h file
2. **Add** the following date and time fields to the configuration `typedef` (to to the location shown below):

```
EFI_HII_TIME  Time;
EFI_HII_DATE  Date;
```

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

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

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

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

#string STR_MY_DATE_PROMPT              #language en "My System Date"

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

6). **Save** MyWizardDriver.uni

7). **Update** the MyWizardDriver.vfr file

8). **Add** the " `goto` " VFR item to allow browser to ender another form by adding the following code before the " `endform` " at approx. line 114

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

```
108     resetbutton
109       defaultstore = MyStandardDefault,
110       prompt   = STRING_TOKEN(STR_STANDARD_DEFAULT_PROMPT_RESET),
111       help     = STRING_TOKEN(STR_STANDARD_DEFAULT_HELP),
112     endresetbutton;
113
114    grayoutif  ideqval MWD_IfrNVData.MyWizardDriverChooseToEnable == 0x0;
115         goto 2,
116       prompt = STRING_TOKEN(STR_GOTO_FORM2), //SecondSetupPage
117       help   = STRING_TOKEN(STR_GOTO_HELP);
118    endif;
119
120   endform;
121
```

9). **Add** the following code between " `endform` " at approx. line 120 and " `endformset` " (the code continues for few pages in this lab guide):

```
// Begin code

    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

    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


    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;
```

```
// End code
```

10). **Save** MyWizardDriver.vfr

Build and test MyWizardDriver

1. At the Terminal Command Prompt (**Cntl-Alt-T**)

```
bash$ cd ~/src/edk2
bash$ build
```

2. **Copy** the `OVMF.fd` BIOS image created from the `build` to the run-ovmf directory naming it `bios.bin`

```
bash$ cp ~/src/edk2/Build/OvmfX64/DEBUG_GCC5/FV/OVMF.fd bios.bin
```

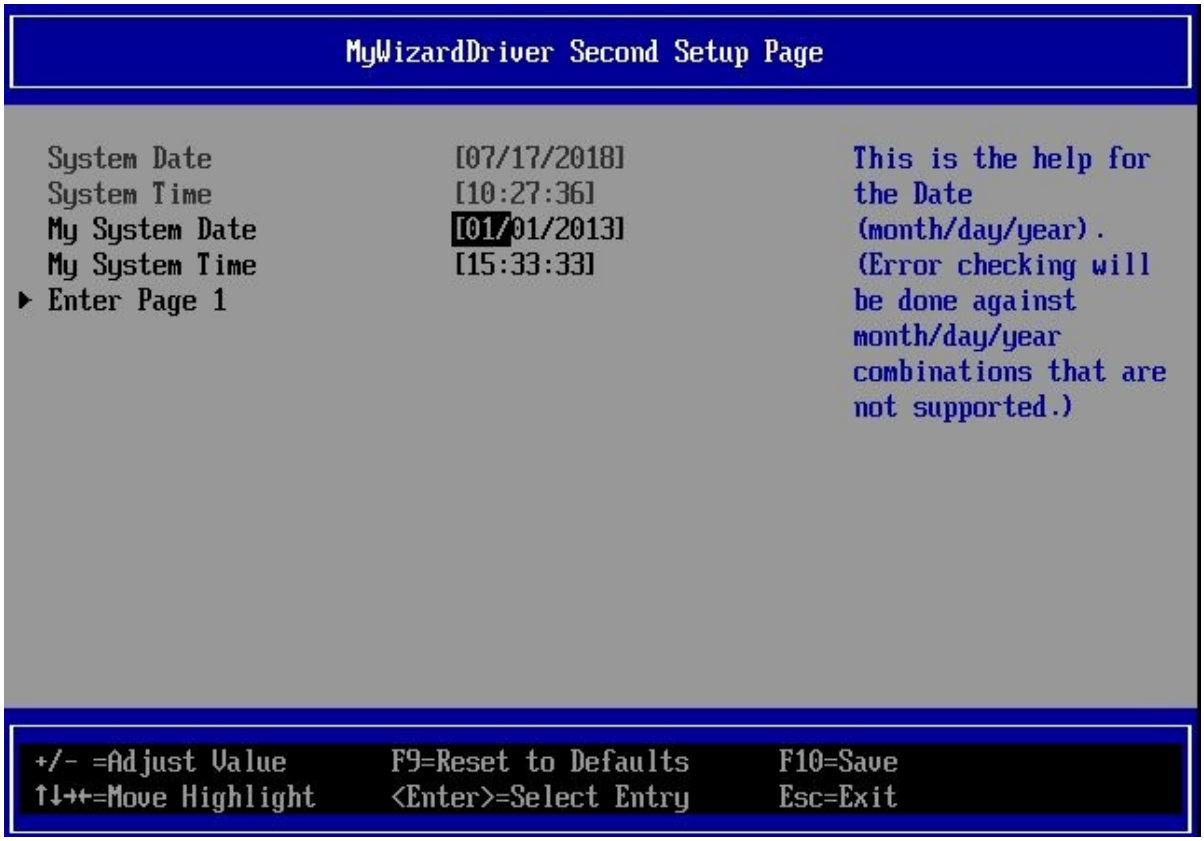
3. **Invoke** Qemu

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

4. At the UEFI Shell prompt,type **exit**
5. Now at the setup front page menu, **select** “Device Manager”
6. Inside the Device Manager menu **press** the down arrow to “My Wizard Driver Sample Formset” **Press** "Enter"

Notice the “Enter Page 2” option. Without `goto` in the `MyWizardDriver.vfr` file, you wouldn’t be able to access page two.

7. **Select** “Enter Page 2” and then **Press** “Enter”
   Notice how the System Date and Time cannot be modified to any other date/time and is grayed out:



8. **Test** by trying to enter the date 02/30/2013, then try a valid leap year date: 02/29/2012.
9. **Press** “Down Arrow” to return to Page 1
10. **Test** the “ `grayoutif”` by going to “Enable My XYZ Device”
11. **Press** the “Spacebar” to toggle off/disable
    Notice the “Select Base Address” , “Name of Configuration” and the “Enter Page 2” fields are now grayed out and not selectable

12. **Press** “Space bar” again to Enable
13. **Press** “F10” then “Escape” to save and exit
14. **Press** “Escape” to exit “Device Manager”
15. **Select** “Continue” and then **Press** "Enter”
16. **Type** “reset” at the Shell prompt

```
FS0:\> reset_
```

17. **Exit** QEMU

For any build issues copy the solution files from ~/FW//LabSolutions/LessonE.10

NOTE: Delete Directory ~/src/edk2/Build/OvmfX64/DEBUG_GCC5/X64/MyWizardDriver before the Build command to build the MyWizardDriver Clean

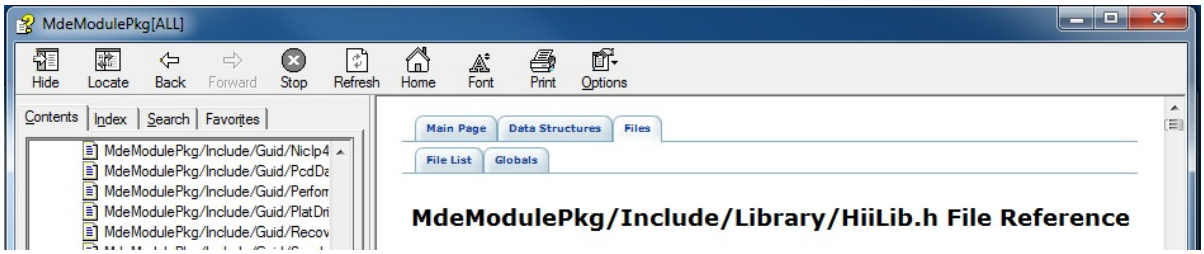
End of Lab 10

Lab 11. Adding Communication from Driver to Console through HII

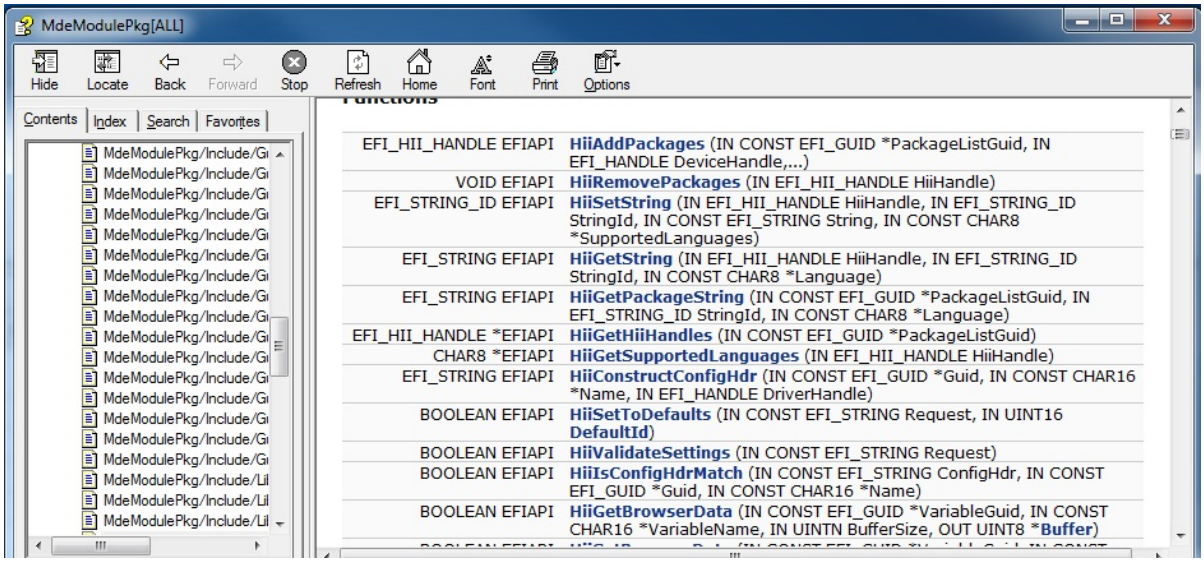
In this lab, you'll add communication from the driver to the console through HII. More specifically, you'll add code to retrieve a string from the HII database and print the string to the console. Then, you'll add the string in French, change the language, and test to ensure the correct language is displayed. 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.

Examine the MdeModulePkg.Chm for HII

1. Use the .Chm reader and open the MdeModulePkg.Chm. For HII database library functions are in the MdeModulePkg .Chm file. Search on HiiLib.h functions
2. Select the Index tab
3. Type: HiiLib.h



Note: Notice the list of Hii function calls available. To get Strings the `HiiGetString` function can be used.



Add changes to your Driver

1. **Update** the ~/src/edk2/OvmfPkg/OvmfPkgX64.fdf file
2. Make your driver stand alone again. **Remove** (or comment out) the include statement in the OvmfPkgX64.fdf file:

```
#INF MyWizardDriver/MyWizardDriver.inf
INF  MdeModulePkg/Universal/Network/IScsiD:
#INF  MyWizardDriver/MyWizardDriver.inf
!if $(BUILD_NEW_SHELL) == TRUE
INF  ShellPkg/Application/Shell/Shell.inf
!endif
```

3. **Save** OvmfPkgX64.fdf
4. **Update** the MyWizardDriver.uni file
5. Add the following code to the top of the file at approx. line 14 as shown:

```
#langdef fr-FR "Francais"
12
13 #langdef en "English"
14 #langdef fr-FR  "Francais"
15
16 #string STR_SAMPLE_FORM_SET_TITLE      #language en  "My Wizard I
17 #string STR_SAMPLE_FORM_SET_HELP       #language en  "Help for Sa
18 #string STR_SAMPLE_FORM1_TITLE         #language en  "My Wizard I
19
```

6. **Add** the following code to the end of the file:

```
#string STR_LANGUAGE_TEST_STRING        #language en "Laurie's Test String"

                                        #language fr-FR "Chaîne de test de Laurie"
```

```
75
76 #string STR_MY_TIME_PROMPT            #language en "My System Time"
77
78 #string STR_LANGUAGE_TEST_STRING       #language en "Laurie's Test String"
79                                       #language fr-FR "Chaîne de test de Laurie"
80
```

7). **Save** MyWizardDriver.uni

8). **Update** the MyWizardDriver.c file

9). **Add** the following local variable for `StringPtr` after " `BOOLEAN ActionFlag;` " and before " `Status = EFI_SUCCESS;` "(as shown below):

```
EFI_STRING StringPtr;
```

```
189   UINTN                           BufferSize;
190   MYWIZARDDRIVER_CONFIGURATION    *Configuration;
191   BOOLEAN                         ActionFlag;
192   EFI_STRING                      StringPtr;
193   Status = EFI_SUCCESS;
194
```

10). Add the following code after " `FreePool (ConfigRequestHdr);` " (as shown below) to edit the driver's entry point with a debug and print statement by making a call to the `HiiGetString` for the token to print (at approx line 364):

```
StringPtr        = HiiGetString (HiiHandle[0], STRING_TOKEN (STR_LANGUAGE_TEST_STRING)
, NULL);
   DEBUG ((EFI_D_INFO,"[MyWizardDriver-Entrypoint] My String was: %s\n", StringPtr) );
   Print(L"%s\n", StringPtr );
```

```
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
```

11). **Save** the MyWizardDriver.c

Build and test Driver

1. At the Terminal Command Prompt (**Cntl-Alt-T**)

```
bash$ cd ~/src/edk2
bash$ build
```

2. **Copy** the `OVMF.fd` BIOS image created from the `build` to the run-ovmf directory naming it `bios.bin`

```
bash$ cp ~/src/edk2/Build/OvmfX64/DEBUG_GCC5/FV/OVMF.fd bios.bin
```

3. **Invoke** Qemu

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

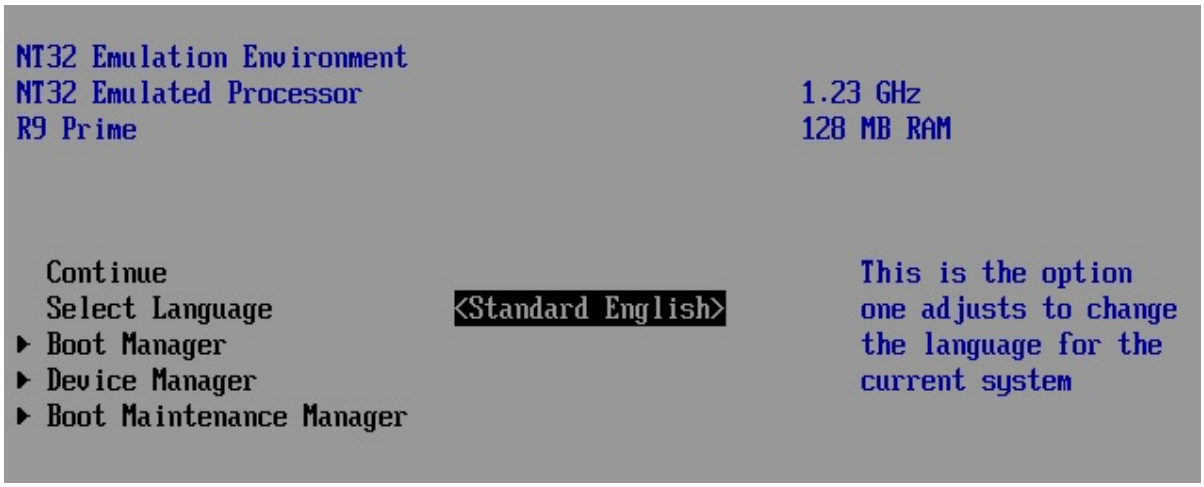
4. **At the UEFI Shell prompt,type** fs0:
5. **Type** Load MyWizardDriver.efi and then **Press** “Enter”

```
Shell> fs0:
FS0:\> load MyWizardDriver.efi
Laurie's Test String
Image 'FS0:\MyWizardDriver.efi' loaded at 604A000 - Success
FS0:\> _
```

Notice that the string’s English version is displayed:

6. **Type** Reset and then **Press** “Enter”
7. **Type** "build run" and then **Press** “Enter”
8. **Type** "exit" at the shell prompt

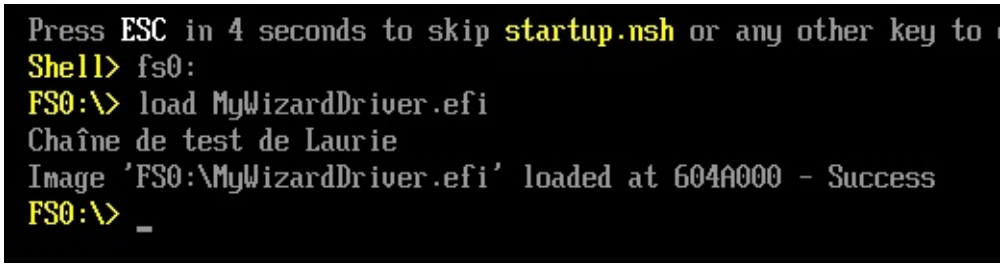
9. **Select Language** and then **Press** “Enter”



10. **Select** “Français” and then **Press** “Enter”

11. **Select** “Continuer” and then **Press** “Enter”

12. At the Shell Prompt, **type** Fs0:
13. **Type** load MyWizardDriver.efi



14. **Type** “reset” and then **Press** “Enter”

15. **Exit** QEMU

For any build issues copy the solution files from ~/FW/LabSolutions/LessonE.11 NOTE: Delete Directory

~/src/edk2/Build/OvmfX64/DEBUG_GCC5/X64/MyWizardDriver before the Build command to build the MyWizardDriver Clean. Make sure you update OvmfPkgX64.fdf.

End Lab 11

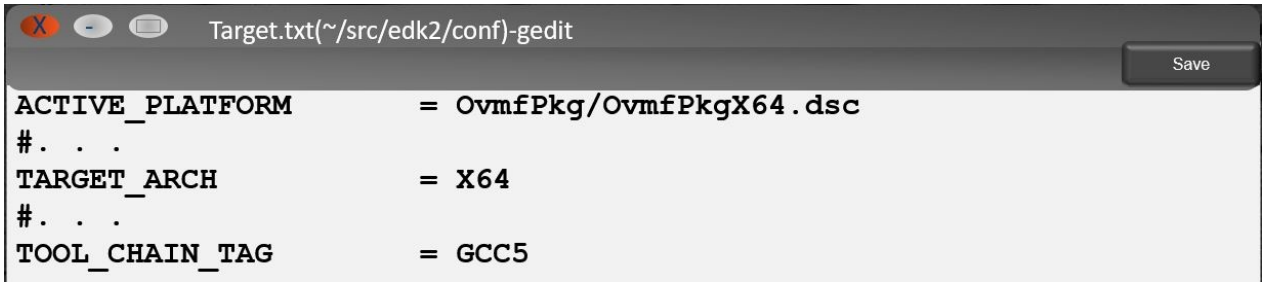
LAB SETUP LINUX FOR UBUNTU 16.04

1. Download the UEFI Training Materials .zip (accept any security notifications)
2. **Click** “Open” and Extract the file to HOME which will take a few minutes
   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. The Default will be in `/FW/...` Directory
   - `~/Lab_Material_FW/FW`
     - Documentation
     - DriverWizard
     - edk2
     - edk2Linux
     - LabSampleCode
3. **Install** the Ubuntu Linux tools:

```
bash$ sudo apt-get install build-essential uuid-dev iasl git
bash$ sudo apt-get install gcc-5 nasm
bash$ sudo apt-get install qemu
```

4. **Create** a directory “src”
   `bash$ mkdir ~src`
5. From the `~.../FW` folder, copy and paste folder “ `~.../FW/edk2` ” to `~src`
6. **Rename** or **mv** the directory “ `~src/edk2/BaseTools` ” to something else
   `bash$ cd ~src/edk2`
   `bash$ mv BaseTools BaseToolsX`
7. **Extract** the file `~FW/edk2Linux/BaseTools.tar.gz` to `~src/edk2`
   `bash$ cd ~src/edk2`
8. Make the BaseTools and setup the environment
   `bash$ make –C BaseTools`
   `bash$ . edksetup.sh`
9. **Open** the file Conf/target.txt
   `bash$ gedit Conf/target.txt`
10. Update the following:

```
ACTIVE_PLATFORM       = OvmfPkg/OvmfPkgX64.dsc
TARGET_ARCH           = X64
TOOL_CHAIN_TAG        = GCC5
```



1. **Save** and Exit target.txt
2. To build OvmfPkg **Type**

   ```
   bash$ build
   ```
3. The file OVMF.fd should be in the Build directory:

   ```
   ~/src/edk2/Build/OvmfX64/DEBUG_GCC5/FV/OVMF.fd
   ```

Build errors

For build errors the Build option for GCC5 may need to be updated:

1. **Edit** ~/src/edk2/tools_def.txt
2. **Find** `DEFINE GCC44_ALL_CC_FLAGS` remove the " `-Werror` " flag because it will treat warnings as errors.

   ```
   DEFINE GCC44_ALL_CC_FLAGS = -g -fshort-wchar -fno-builtin -fno-strict-aliasing -Wall
   -Wno-array-bounds -ffunction-sections -fdata-sections -include AutoGen.h -fno-common -
   DSTRING_ARRAY_NAME=$(BASE_NAME)Strings
   ```
3. **Save** tools_def.txt
4. Build again **Type**

   ```
   bash$ build
   ```

Invoke QEMU to run UEFI Shell

1. **Create** a run-ovmf directory under the home directory

   ```
   bash$ cd ~
   bash$ mkdir ~run-ovmf
   bash$ cd run-ovmf
   ```
2. **Create** a directory hda-contents to use as a hard disk image

   ```
   bash$ mkdir hda-contents
   ```
3. **Copy** the `OVMF.fd` BIOS image created from the `build` to the run-ovmf directory naming it `bios.bin`

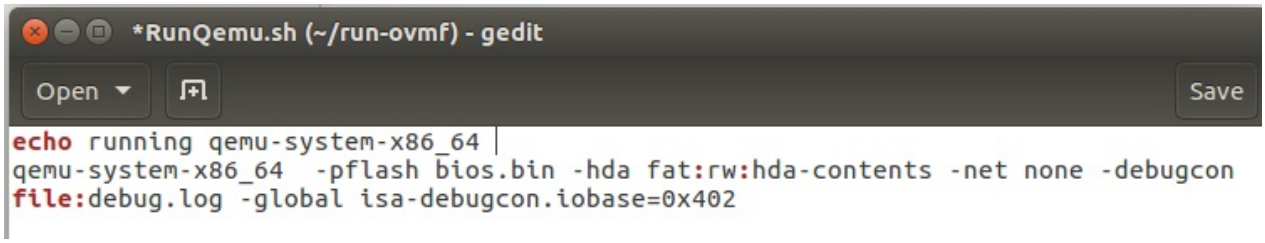
```
bash$ cp ~/src/edk2/Build/OvmfX64/DEBUG_GCC5/FV/OVMF.fd bios.bin
```

4. Create a Linux shell script to run the QEMU from the run-ovmf directory

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

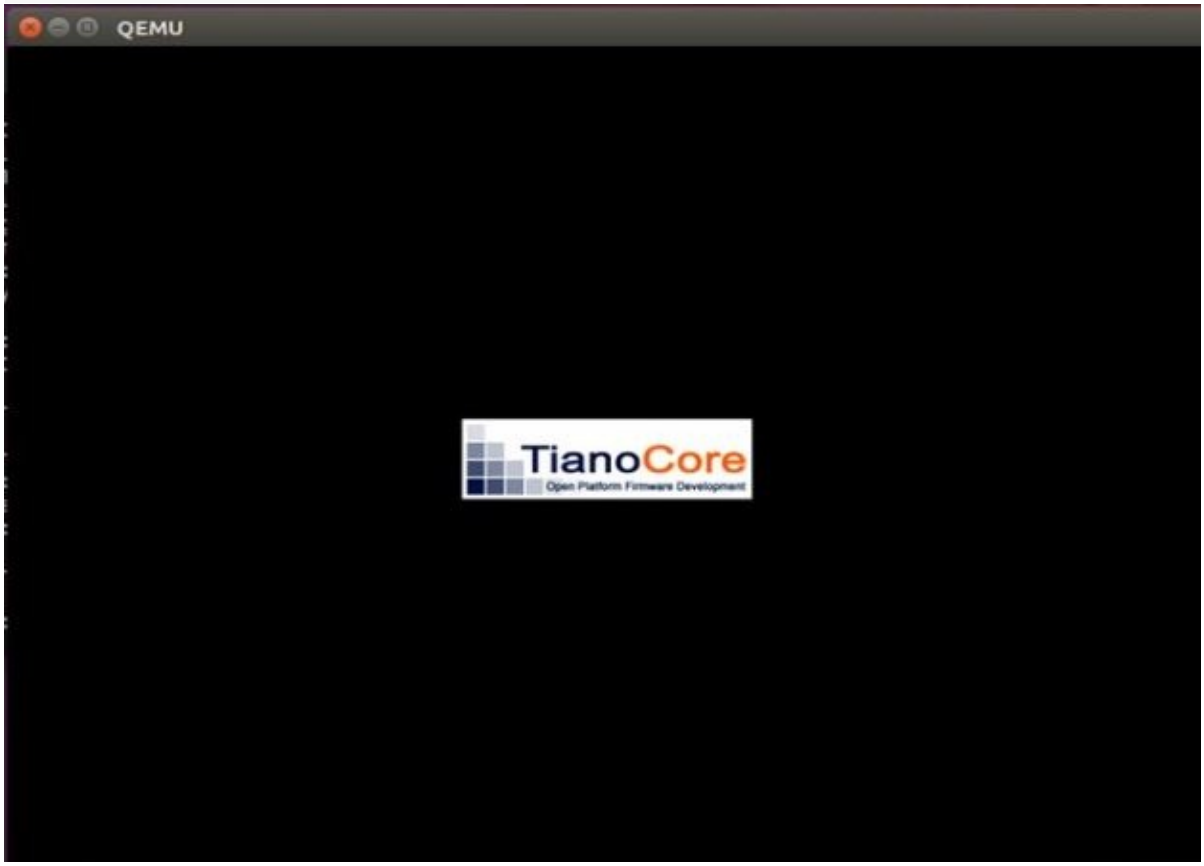
   Add the following:

```
qemu-system-x86_64 -pflash bios.bin -hda fat:rw:hda-contents -net none -debugcon file:
debug.log -global isa-debugcon.iobase=0x402
```



1. **Save** RunQemu.sh and exit
2. **Run** the RunQemu.sh Linux Shell Script:

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



End of Lab Setup for Linux

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