# Update your USB Library to support new HID devices

This section details how to build support for additional device types. This uses usbdhidcustom.c which was designed to minimize the code edit requirements when prototyping a new device.

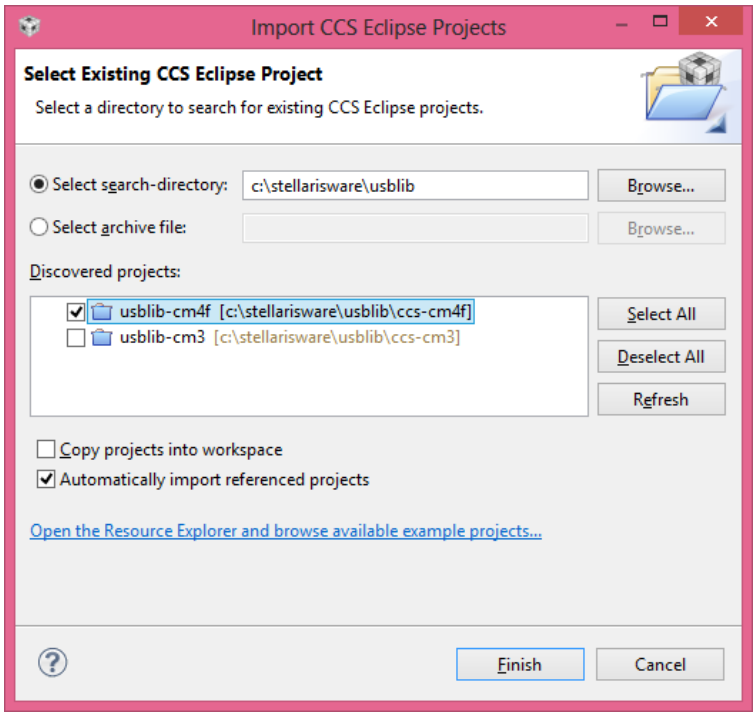
Prerequisites:

- The full Stellarisware package is installed, these instructions assume its installed to C:\Stellarisware
- You have the files from this distribution, including usbdhidcustom.h and usbdhidcustom.c, an updated version of usbdhid.c and usb-ids.h and the example projects that utilize this framework (gamepad, keyboard+mouse and volume control)

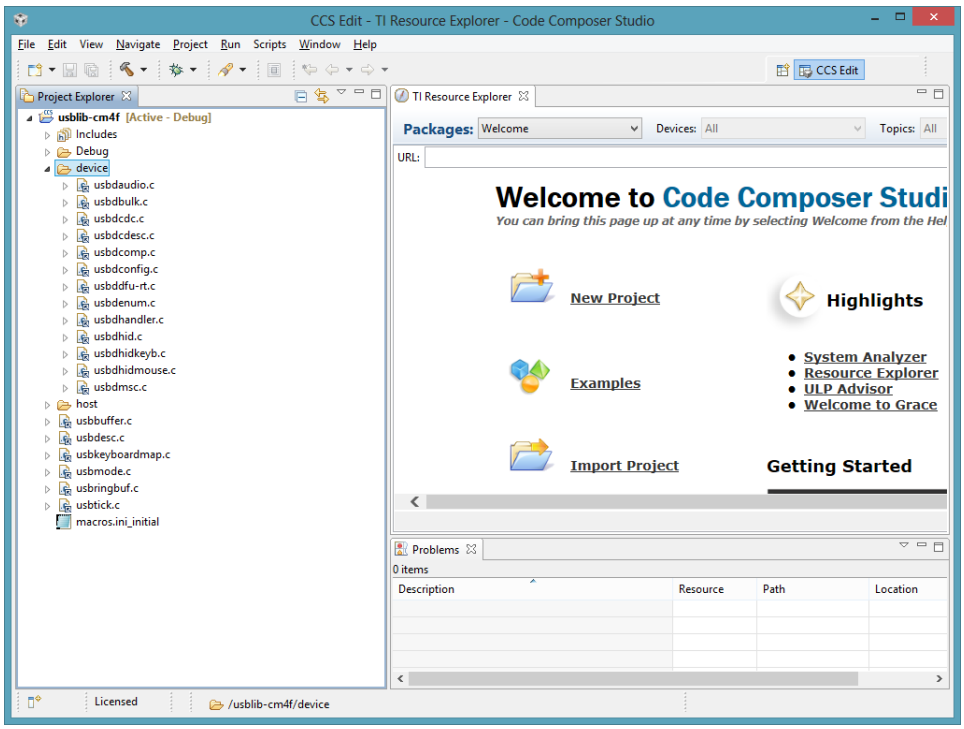
## Steps

**NOTE**: Some steps replace TI header files with updated ones, this is based on the 9453 USB library, if you have a future version you should manually merge the updated definitions to validate there is no conflict.

1. Copy usbdhidcustom.h and usbdhidcustom.h files to c:\stellarisware\usblib\device
2. Copy the updated usbdhid.h to c:\stellarisware\usblib, when prompted to override select Yes. (See note above)
3. Copy the updated usb-ids.h to c:\stellarisware\usblib, when prompted to override select Yes. (See note above)
4. Open Code Composer Studio
5. Create a new workspace (File-> Switch Workspace->Other-> ex. c:\workspace)
6. Import the USB library (Project->Import Existing CCS Eclipse Project).Browse to C:\stellarisware\usblib. Ensure that copy projects into workspace is not selected and automatically import referenced projects is. I have a Launchpad (CM4F) device so I did not modify the CM3 project in these steps.

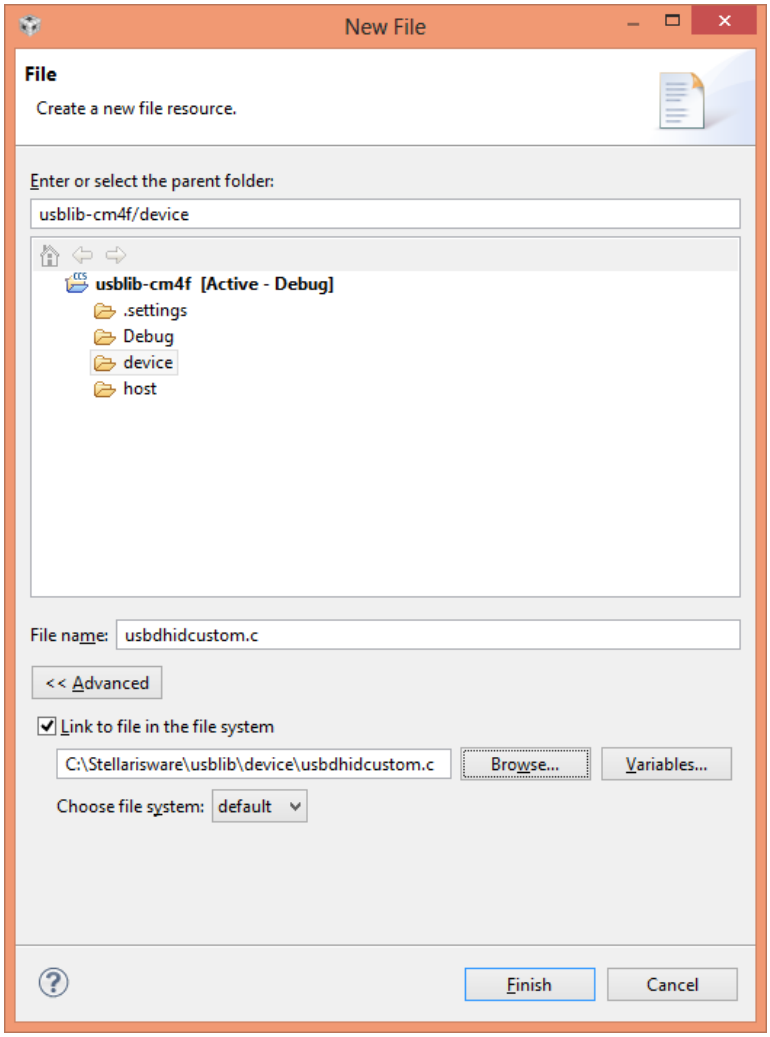



7. Select the left pane and expand the device subfolder.
**NOTE**: Sometimes it doesn't seem to display the devices and host folders, if this occurs recopy the c:\stellarisware\usblib\ccs-cm4f file from the Stellarisware download and run through the instructions again.

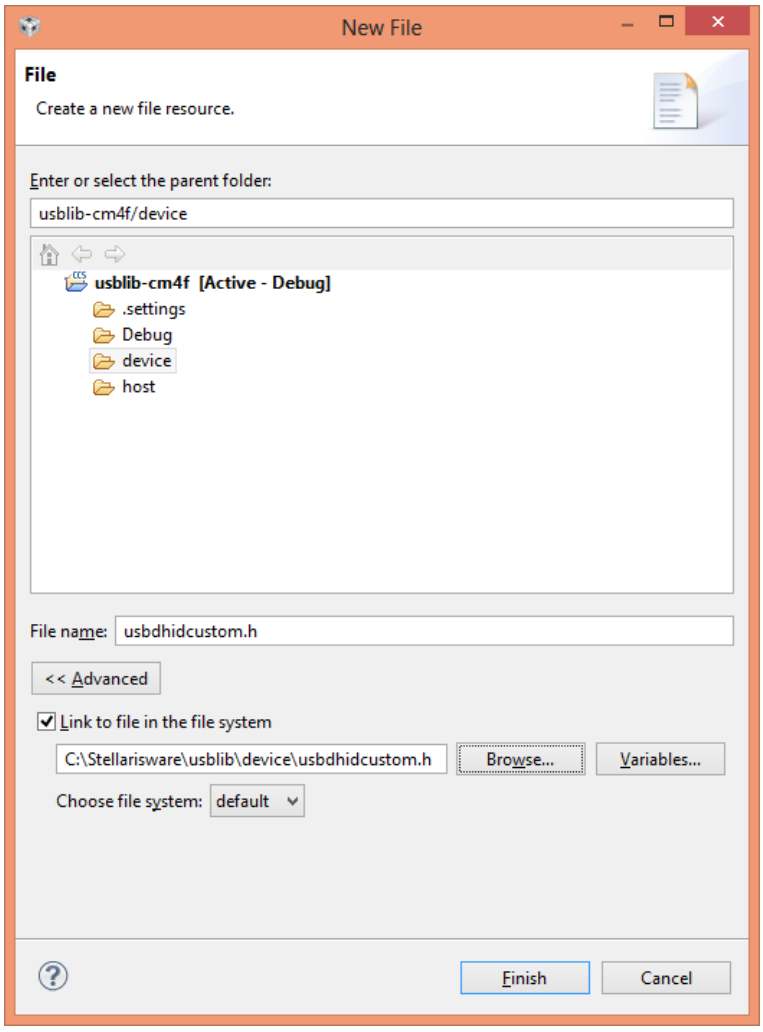



8. Right click on Device and select New File then expand **advanced** and select **Link.** Specify the path to usbdhidcustom.c (Copied in step 1). You should see the file appear in the left pane with an arrow

over the filename indicating it's a linked file




9. Right click on Device and select New File then expand **advanced** and select **Link.** Specify the path to usbdhidcustom.h (Copied in step 1). You should see the file appear in the left pane with an arrow over the filename indicating it's a linked file



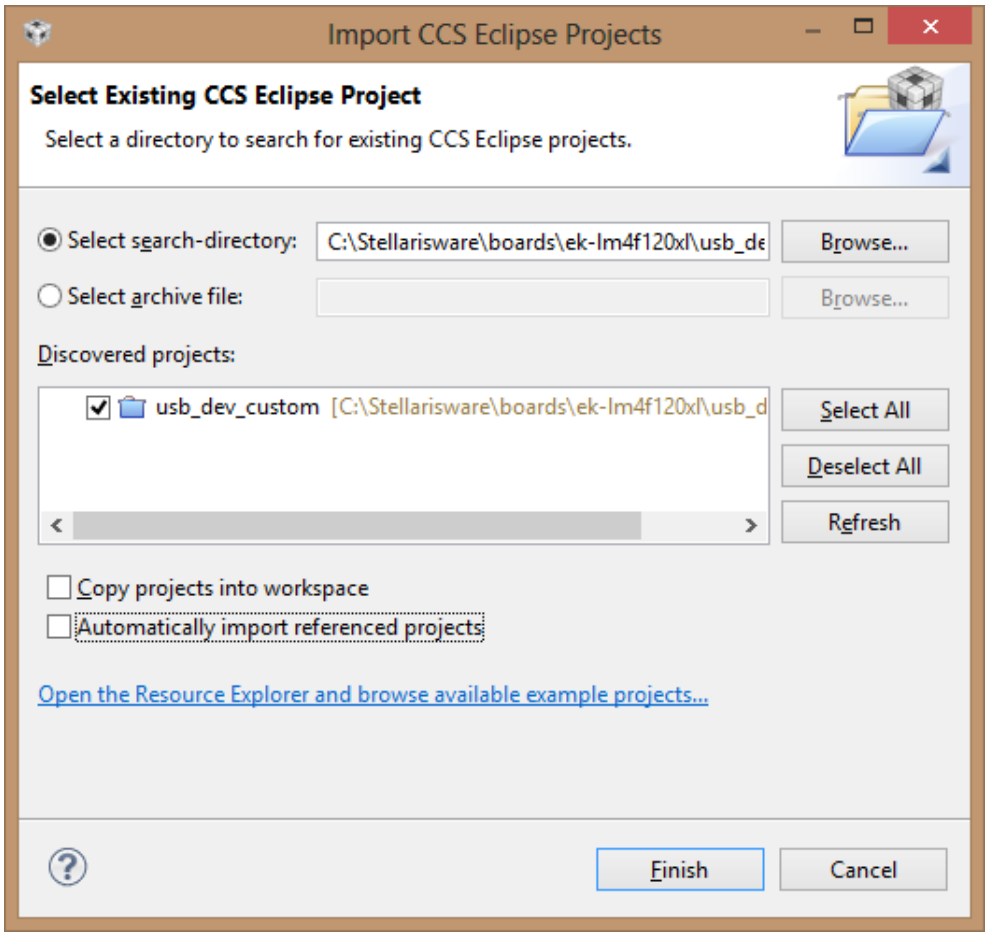

10. Select Project -> Build Project. This will build USB support including the usbdhidcustom device components with the sample device descriptor included in the file. Each time you build a new device (covered in the following examples) you will need to adjust the descriptor and rebuild the USB library. The next pages will take you through using this library to build 3 devices not supported by the vanilla TI provided files (A simple HID volume control, a gamepad and a keyboard + mouse)

# Build a simple HID volume control based on the Stellaris Launchpad and usbdhidcustom.c

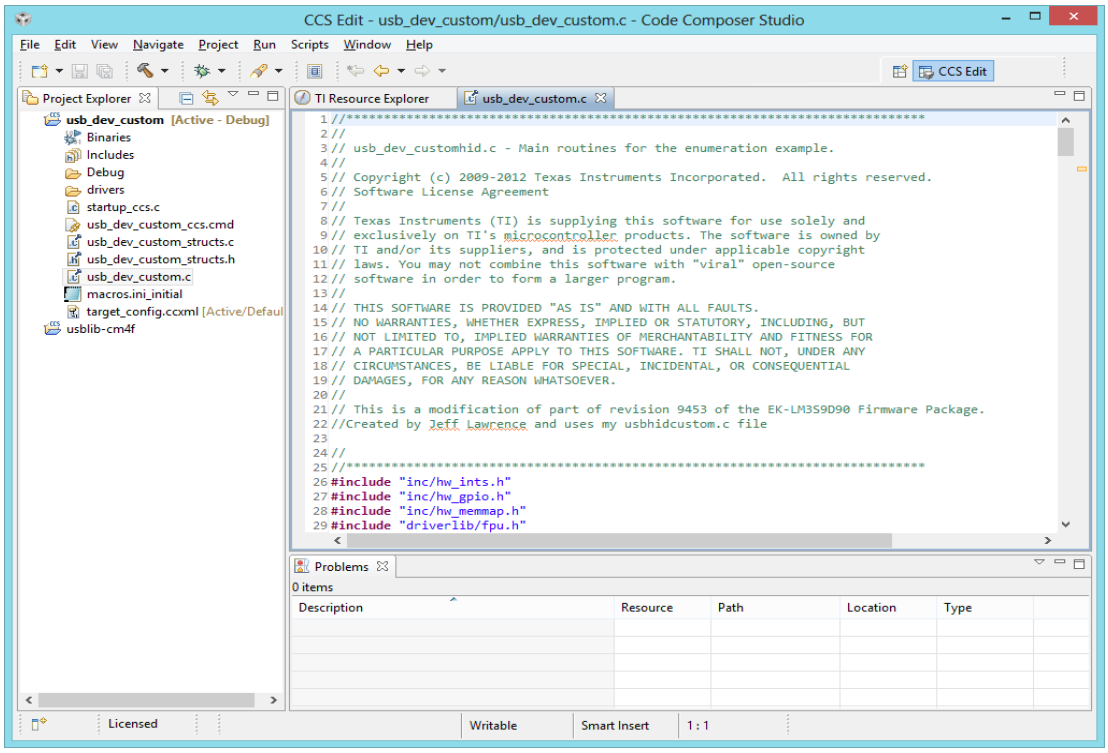
## Prerequisites

- A USB library built with support for usbdhidcustom.c and the updated usbdhid.h and usb-ids.h file (See last section)
- Example project usb_dev_custom
- Stellaris Launchpad board

1) Copy the usb_dev_custom project to C:\Stellarisware\boards\ek-lm4f120xl\usb_dev_custom
2) Utilize the workspace from the last example or open a new workspace and import the usblib-cmf4 project as described in the last section.
3) Import the usb_dev_custom project (Project->Import Existing CCS Eclipse Project). Browse to C:\Stellarisware\boards\ek-lm4f120xl\usb_dev_custom. Ensure Copy projects into workspace is not selected




4) In the left pane expand the newly imported project and select the file usb_dev_custom.c




5) In the right pane scroll down to line 54 where you will find the descriptor for a volume control device.

```
UsagePage(USB_HID_CONSUMER_DEVICE),
Usage(USB_HID_USAGE_CONSUMER_CONTROL),
Collection(USB_HID_APPLICATION),
       LogicalMinimum(0),
    LogicalMaximum(1),
    Usage(USB_HID_VOLUME_UP),
    Usage(USB_HID_VOLUME_DOWN),
    ReportSize(1),
    ReportCount(2),
    Input(USB_HID_INPUT_DATA | USB_HID_INPUT_VARIABLE | USB_HID_INPUT_RELATIVE),
    ReportCount(6),
    Input(USB_HID_INPUT_CONSTANT | USB_HID_INPUT_ARRAY | USB_HID_INPUT_ABS),
EndCollection,
```

6) This descriptor indicates that a volume device sends 1 byte in its HID report - 2 bits of that byte are used to represent the volume up/down, and 6 bits are sent as padding.
7) Select the entire **static const unsigned char g_pucCustomHidReportDescriptor[]** definition (which includes the descriptor) and select Copy (or CTRL-C)
8) Expand the usblib-cm4f project and under device select usbdhidcustom.c (which we added earlier). Find the defintion for **g_pucCustomHidReportDescriptor[]** and replace with the one from the volume control sample.
9) Since you have modified the descriptor you also need to edit the header to reflect the new size of that descriptor. Right click in the usbdhidcustom.c file and select Toggle Source/Header. Find the line that `#define CUSTOMHID_REPORT_SIZE` and set the value to 1 to reflect the 1 byte descriptor for the volume control.
10) Select File->Save All

11) Make sure in the left pane usblib-cm4f is highlighted and then select Project->Build Project. **(Each time you modify a descriptor you must update usbdhidcustom.c and .h and rebuild)**
12) Go back to usb_dev_custom and find the ***CustomHidChangeHandler*** function.

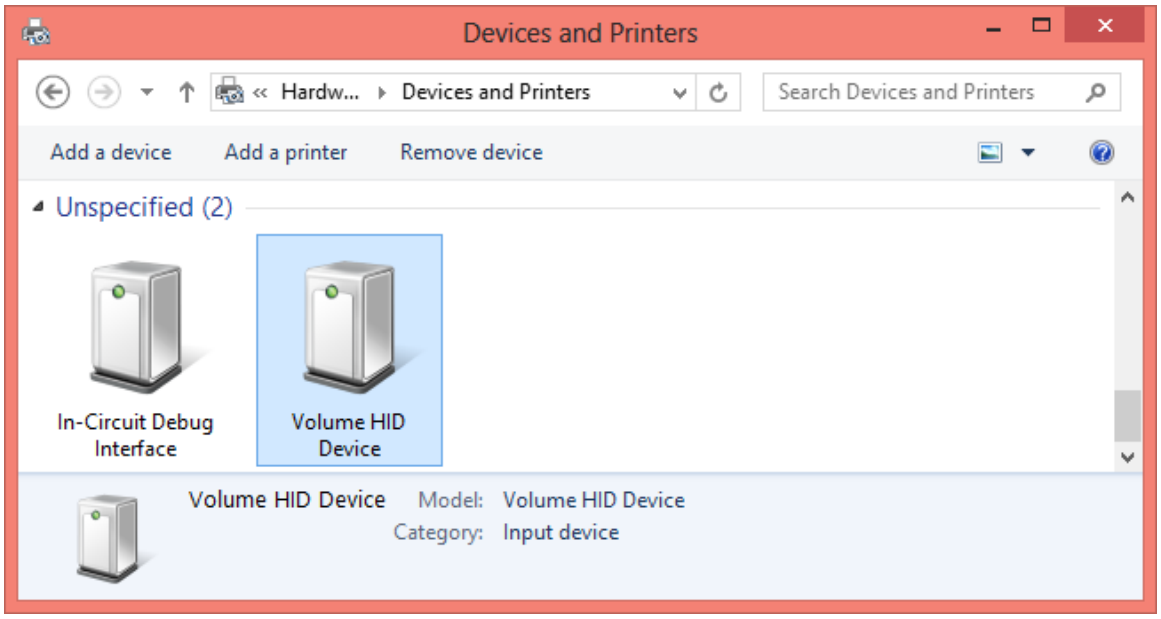
    **Things to note:**

    A char array named Devicereport is used to pass the device report values. In this example you only need to send a single byte value representing the volume (the 2 relevant bits up/down and the 6 bits for padding). The **USBHIDCustomHidStateChange** call:

    ```
    ulRetcode = USBDHIDCustomHidStateChange((void *)&g_sCustomHidDevice, 0,
    DeviceReport)
    ```

    Has a second argument set at **0**, this is for devices with multiple reportIDs, since the volume one doesn't require this it's set to 0. **Note**: The reason for the array and reportID being included is to simplify compatibility with more complex devices as provided in the following examples.

13) Build and deploy this project and you should see a new device appear in Devices (Win8 shown) and the Launchpad LED should light solid green to indicate a connected device.




14) Press the left and right buttons should change the volume. You may need to adjust the systick interrupt if the control is too sensitive.

**Troubleshooting**: If you have issues try the following:

- Verify the descriptor from usb_dev_custom was correctly copied to usbdhidcustom.c
- Verify that usbdhidcustom.h has `CUSTOMHID_REPORT_SIZE` set to 1
- Clean and build the USB library and ensure there are no build errors
- Clean and build the usb_dev_custom project and deploy.

## Build a Gamepad based on the Stellaris Launchpad

This example updates the code to act as a gamepad instead. This demonstrates the minimal changes required to create a different HID device.

1) Edit usbdhidcustom.c and replace the volume control descriptor with this one. This defines a basic gamepad with 16 buttons and 2 joypads (X,Y represent 1 and Rx,Z represent another). This is a 6 byte descriptor with no required padding. (2 bytes for buttons and 1 for each of the 4 axis)

```
static const unsigned char g_pucCustomHidReportDescriptor[]=
{
  UsagePage(USB_HID_GENERIC_DESKTOP),
  Usage(USB_HID_GAMEPAD),
  Collection(USB_HID_APPLICATION),
    Collection(USB_HID_PHYSICAL),
    //
    // 8 - 1 bit values for the first set of buttons.
    //
    UsagePage(USB_HID_BUTTONS),
    UsageMinimum(1),
    UsageMaximum(8),
    LogicalMinimum(0),
    LogicalMaximum(1),
    ReportSize(1),
    ReportCount(8),
    Input(USB_HID_INPUT_DATA | USB_HID_INPUT_VARIABLE | USB_HID_INPUT_ABS),

    //
    // 8 - 1 bit values for the second set of buttons.
    //
    UsagePage(USB_HID_BUTTONS),
    UsageMinimum(1),
    UsageMaximum(8),
    LogicalMinimum(0),
    LogicalMaximum(1),
    ReportSize(1),
    ReportCount(8),
    Input(USB_HID_INPUT_DATA | USB_HID_INPUT_VARIABLE | USB_HID_INPUT_ABS),

    //
    // The X, Y Z and Rx (Will appear as two thumb controls.
    //
    UsagePage(USB_HID_GENERIC_DESKTOP),
    Usage(USB_HID_X),
    Usage(USB_HID_Y),
    Usage(USB_HID_Z),
    Usage(USB_HID_RX),
    LogicalMinimum(-127),
    LogicalMaximum(127),
    ReportSize(8),
    ReportCount(4),
    Input(USB_HID_INPUT_DATA | USB_HID_INPUT_VARIABLE | USB_HID_INPUT_ABS),
```

```
    EndCollection,
  EndCollection,
};
```

2) Edit usbdhidcustom.h and update the size to 6 bytes to reflect the 6 bytes required in this HID report.

```
#define CUSTOMHID_REPORT_SIZE 6
```

3) Within ***CustomHidChangeHandler*** Change the Devicereport declaration to reflect the additional values sent in the report

```
signed char DeviceReport[6];
DeviceReport[0]=0; //First 8 buttons
DeviceReport[1]=0; //Second 8 buttons
DeviceReport[2]=0; //X
DeviceReport[3]=0; //Y
DeviceReport[4]=0; //Z
DeviceReport[5]=0; //Rx
```

4) Update the button code (Launchpad specific) to send different gamepad data depending on the button being pressed. The required values aren't critical but the provided examples help make it clear which buttons are being pressed.

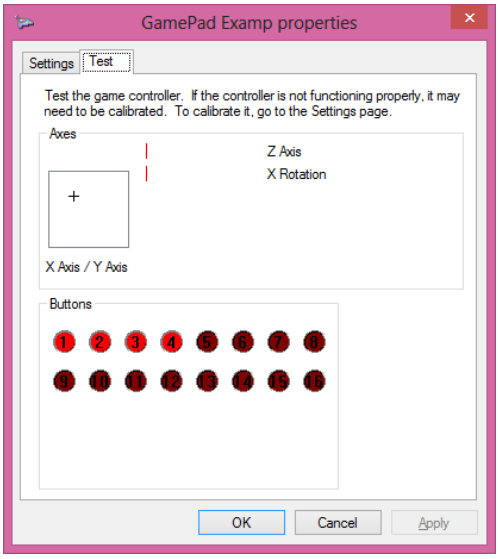
```
//
//Read Left button and press left buttons and move to the left
//
ulButton = ROM_GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_4);
if(ulButton == 0)
{
   DeviceReport[0]=0x0F;
   DeviceReport[1]=0;
   DeviceReport[2]=-60;
   DeviceReport[3]=-60;
   DeviceReport[4]=-125;
   DeviceReport[5]=-125;
}
//
//Read Right button and press right buttons and move to the right
//
ulButton2 = ROM_GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_0);
if(ulButton2 == 0)
{
   DeviceReport[0]=0;
   DeviceReport[1]=0xF0;
   DeviceReport[2]=60;
   DeviceReport[3]=60;
   DeviceReport[4]=125;
   DeviceReport[5]=125;
}
```

5) Change the PID in usb_dev_custom_structs.c from **USB_PID_CUSTOM** to **USB_PID_GAMEPAD**
6) (Optional) Update the product string to "Gamepad Example" so that the device manager name makes sense.

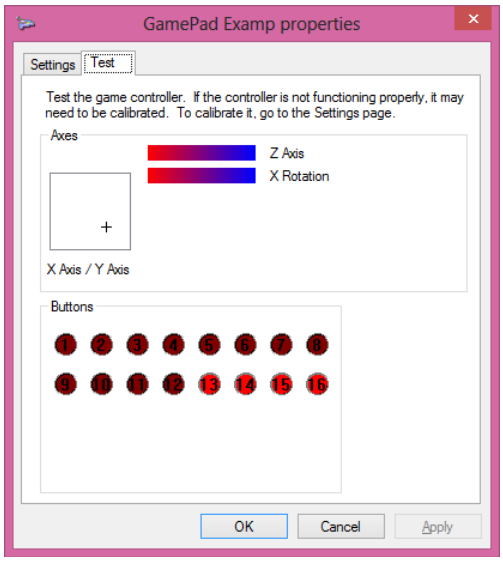
```
const unsigned char g_pProductString[] =
{
  (15 + 1) * 2,
  USB_DTYPE_STRING,
  'G', 0, 'a', 0, 'm', 0, 'e', 0, 'p', 0, 'a', 0, 'd', 0, ' ', 0, 'E', 0,  'x', 0, 'a', 0, 'm', 0, 'p', 0, 'l', 0, 'e', 0
};
```

7) Build both the USB project and the custom project and deploy to your Launchpad
8) Windows should detect and install drivers for a gamepad


9) If you right click on the gamepad and select settings you press the Launchpad buttons and validate the response
    a. Left Button Pressed




    b. Right Button (The Z axes should match the X but I had goofed something up in the calibration)




10) (Optional) To add a Point of View Hat to the gamepad. Modify the HID report descriptor to also include the POV values (4 bits)

    Usage(USB_HID_HAT),
    LogicalMinimum(0),
    LogicalMaximum(3),
    UnitRotation_deg,
    ReportSize(4),
    ReportCount(1),
    Input(USB_HID_INPUT_DATA | USB_HID_INPUT_VARIABLE | USB_HID_INPUT_ABS),

    Since all HID reports must be in byte chunks you need to add an additional 4 bits of padding

    ReportSize(4),
    ReportCount(1),
    Input(USB_HID_INPUT_CONSTANT | USB_HID_INPUT_ARRAY | USB_HID_INPUT_ABS),

11) Update **`CUSTOMHID_REPORT_SIZE to 7`** `to reflect the additional byte`
12) Update DeviceReport to a 7 byte char[] and pass the HAT Value in the last byte. You can use value 1 and 3 as example values.

# Build a Keyboard + Mouse HID device based on the Stellaris Launchpad

This example updates the code to act as keyboard and mouse combination device. The composite device examples provided by TI do not support a combination HID device, but the flexibility of the HID standard allows a single descriptor to define multiple HID devices. Notice that each section begins with a ReportID byte - this allows the host to map a sent report to the correct device. These examples rely on all reports starting with 1 (0 represents a device without a reportid in the code)

1) Edit usbdhidcustom.c and replace the existing report descriptor with this one

```
static const unsigned char g_pucCustomHidReportDescriptor[]=
{
 UsagePage(USB_HID_GENERIC_DESKTOP),
   Usage(USB_HID_KEYBOARD),
   Collection(USB_HID_APPLICATION),

     // Report ID (One byte)
     ReportID(1),

     //
     // Modifier keys.
     // 8 - 1 bit values indicating the modifier keys (ctrl, shift...)
     //
      ReportSize(1),
      ReportCount(8),
      UsagePage(USB_HID_USAGE_KEYCODES),
      UsageMinimum(224),
      UsageMaximum(231),
      LogicalMinimum(0),
      LogicalMaximum(1),
      Input(USB_HID_INPUT_DATA | USB_HID_INPUT_VARIABLE | USB_HID_INPUT_ABS),

      //
      // One byte of rsvd data required by HID spec.
      //
      ReportCount(1),
      ReportSize(8),
      Input(USB_HID_INPUT_CONSTANT),

      //
      // Keyboard LEDs.
      // 5 - 1 bit values.
      //
      ReportCount(5),
      ReportSize(1),
      UsagePage(USB_HID_USAGE_LEDS),
      UsageMinimum(1),
      UsageMaximum(5),
      Output(USB_HID_OUTPUT_DATA | USB_HID_OUTPUT_VARIABLE |  USB_HID_OUTPUT_ABS),

      //
      // 1 - 3 bit value to pad out to a full byte.
```

```
      //
      ReportCount(1),
      ReportSize(3),
      Output(USB_HID_OUTPUT_CONSTANT), //LED report padding
      //
      // The Key buffer.
      // 6 - 8 bit values to store the current key state.
      //
     ReportCount(6),
     ReportSize(8),
     LogicalMinimum(0),
     LogicalMaximum(101),
     UsagePage(USB_HID_USAGE_KEYCODES),
     UsageMinimum (0),
     UsageMaximum (101),
     Input(USB_HID_INPUT_DATA | USB_HID_INPUT_ARRAY),
 EndCollection,

UsagePage(USB_HID_GENERIC_DESKTOP),
 Usage(USB_HID_MOUSE),
 Collection(USB_HID_APPLICATION),
 Usage(USB_HID_POINTER),
 Collection(USB_HID_PHYSICAL),

 // Report ID (One byte)
ReportID(2),
//
// The buttons.
//
UsagePage(USB_HID_BUTTONS),
UsageMinimum(1),
UsageMaximum(3),
LogicalMinimum(0),
LogicalMaximum(1),
//
// 3 - 1 bit values for the buttons.
//
ReportSize(1),
ReportCount(3),
Input(USB_HID_INPUT_DATA | USB_HID_INPUT_VARIABLE | USB_HID_INPUT_ABS),
//
// 1 - 5 bit unused constant value to fill the 8 bits.
//
ReportSize(5),
ReportCount(1),
Input(USB_HID_INPUT_CONSTANT | USB_HID_INPUT_ARRAY | USB_HID_INPUT_ABS),
//
// The X and Y axis.
//
UsagePage(USB_HID_GENERIC_DESKTOP),
Usage(USB_HID_X),
Usage(USB_HID_Y),
LogicalMinimum(-127),
```

```
LogicalMaximum(127),
//
// 2 - 8 bit Values for x and y.
//
ReportSize(8),
ReportCount(2),
Input(USB_HID_INPUT_DATA | USB_HID_INPUT_VARIABLE | USB_HID_INPUT_RELATIVE),
EndCollection,
EndCollection,
};
```

2) Edit usbdhidcustom.h and update the size to 9 bytes. This value reflects the largest individual report identified by the descriptor. In this case that is the keyboard which requires 8 bytes + 1 byte for the descriptor. So use 9

```
#define CUSTOMHID_REPORT_SIZE        9
```

3) Within ***CustomHidChangeHandler*** Replace the Devicereport declaration with a separate keyboard and mouse one (or you could just add a devicereport2 declaration). For clarify I will use keyboard and mouse. Also create two values to hold the reportIDs

```
// Initialize all keys to 0 - this un-presses a key if you press one
//
Int KeyboardReportID=1;
signed char KeyboardReport[8];
KeyboardReport[0]=0; //modifier
KeyboardReport[1]=0; //reserved always 0
KeyboardReport[2]=0; //key1
KeyboardReport[3]=0; //key2
KeyboardReport[4]=0; //key3
KeyboardReport[5]=0; //key4
KeyboardReport[6]=0; //key5
KeyboardReport[7]=0; //key6

//
// Initialize mouse values to 0, mouse movement is incremental so it won't reset the cursor
//
Int MouseReportID=2;
signed char MouseReport[3];
MouseReport[0]=0; //Buttons
MouseReport[1]=0; //X
MouseReport[2]=0; //Y
```

4) Update the button code (Launchpad specific) to send mouse movements or keystrokes depending on the button being pressed.

```
//
//Read Left button and move mouse if pressed
//
ulButton = ROM_GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_4);
```

```
if(ulButton == 0)
{
  MouseReport[0]=0; //Buttons
  MouseReport[1]=-15; //X increment to the left
  MouseReport[2]=-5; //Y increment upwards
}

//
//Read Right button and sent ABCDEF if pressed
//
ulButton2 = ROM_GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_0);
if(ulButton2 == 0)
{
  KeyboardReport[0]=0; //modifier (CTRL, ALT, SHIFT, etc)
  KeyboardReport[1]=0; //Reserved, always equals 0
  KeyboardReport[2]=HID_KEYB_USAGE_A; //key1 -these are HID scan codes (4=a)
  KeyboardReport[3]=HID_KEYB_USAGE_B; //key2 - b
  KeyboardReport[4]=HID_KEYB_USAGE_C; //key3 - c
  KeyboardReport[5]=HID_KEYB_USAGE_D; //key4 - d
  KeyboardReport[6]=HID_KEYB_USAGE_E; //key5 - e
  KeyboardReport[7]=HID_KEYB_USAGE_F; //key6 - f
}
```

5) Since you have 2 different arrays to send make 2 separate calls to **USBDHIDCustomHidStateChange.** One should use they keyboard reportID and array the other the mouse

```
ulRetcode = USBDHIDCustomHidStateChange((void *)&g_sCustomHidDevice, KeyboardReportID, KeyboardReport);
ulRetcode = USBDHIDCustomHidStateChange((void *)&g_sCustomHidDevice, MouseReportID, MouseReport);
```

6) Change the PID in usb_dev_custom_structs.c to USB_PID_CUSTOM
7) (Optional) Update the product string to something more appropriate, in this case Dual HID device

```
const unsigned char g_pProductString[] =
{
  (15 + 1) * 2,
  USB_DTYPE_STRING,
  'D', 0, 'u', 0, 'a', 0, 'l', 0, ' ', 0, 'H', 0, 'I', 0, 'D', 0, ' ', 0,
  'D', 0, 'e', 0, 'v', 0, 'i', 0, 'c', 0, 'e', 0
};
```

8) Save All unsaved changes (File->Save All)
9) Build both the USB project and the custom project and deploy to your Launchpad
10) Windows should detect and install drivers for a Dual HID Device that appears as a keyboard in Devices. **NOTE:** This example reuses the PID of the volume control device, sometimes the OS caches the driver and is confused by this, if that occurs right click on the device and select Remove Device (or uninstall prior) Or you can use a static value instead of USB_PID_CUSTOM to force windows to reinstall the driver.


11) Open notepad and what you should see is that the left Launchpad button jumps the mouse cursor and the right sends ABCDEF to the notepad instance

**Note**: The keyboard handling is very simple; refer to the full keyboard example provided by TI for a full implementation.

The attached files include completed projects as well as some of the standalone examples of mice, keyboards and Composite devices ported from another board to be compatible with the Launchpad.

Jeff Lawrence 1/9/2013
Lawrence_jeff on the forums