USB Serial Device Firmware Upgrade (DFU)

Abstract:

This example illustrates how to add USB Device Firmware Upgrade (DFU) support to an existing TI-RTOS USB device application.

The TI-RTOS USB Serial example for the EK-TM4C1294XL Evaluation Kit is used as the starting point. You will combine this example with a boot loader which is capable of performing an in-field upgrade of the firmware. The boot loader uses the USB DFU Device class to perform the upgrade. You will configure the boot loader to use a GPIO pin to enter upgrade mode. This approach does not require any significant changes to the original example.

Overview

Introduction

Setup **Install required products**.

This section lists the required products and general settings needed to satisfy the prerequisites for this example.

Part 1 **Import USB Serial example**.

You will import the USB Serial Device example from the TI-RTOS product. This example makes your evaluation kit look like a USB Communication Class Device. Once connected to your computer, you can use a terminal emulator to communicate with it.

Part 2 **Build the TivaWare boot loader**.

You will create a CCS project in your workspace to build the TivaWare boot loader. The boot loader source files are available in your TI-RTOS product install, but you must import the source files into your project and configure the boot loader for USB DFU functionality. Once built, you will load and run the boot loader on your evaluation kit.

Part 3 **Combine boot loader with application**.

In order to make the existing example (built in Part 1) compatible with the boot loader, you will modify the example memory map to make room for the boot loader which will be resident in flash memory. You will also create a custom target configuration for both the boot loader and the application.

Part 4 **Firmware upgrade image**.

At this point, you have a working USB device. Let's pretend you have manufactured many devices and they are deployed in the field. In this part, you will make improvements to the application and build a firmware upgrade image. This firmware image will be used to upgrade your devices in the field (i.e. without using CCS).

Part 5 **Firmware upgrade**.

The final step is to actually perform an in-field upgrade of your USB CDC device. You will connect your device to your computer, reboot the device in upgrade mode, then perform the upgrade from your computer using a Windows based utility program. The upgrade will replace the existing firmware on your device with the new firmware image you generated in part 4 above.

Part 6 **USB Composite Device**

This part illustrates how to create a USB Composite device. A completed project is provided. You will import this project into your workspace and then study it. Key elements of the project are discussed in text.

Notes:

A. The example contains a folder named ccs which contains the final versions of the projects. You may import these projects into your workspace and compare them to the results you achieved.

```
File > Import
Code Composer Studio > CCS Projects
Next

Select search-directory: ek_tm4c1294xl_usb_serial_dfu/ccs
Select the projects you wish to import
Copy projects into workspace > Select
Finish
```

B. Linux is not supported at this time. The TivaWare DFU utility programs are only available on Windows.

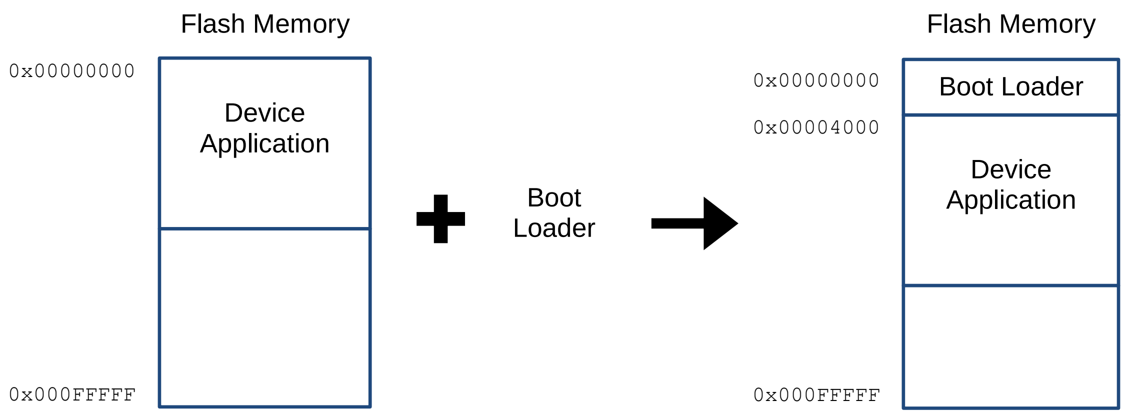
Introduction

This example illustrates how to add USB based firmware upgrade capability to your device application. This means that once your device has been deployed in the field, then end-user has the ability to upgrade the firmware without significant effort. This example will use the USB Device Firmware Upgrade (DFU) protocol to manage the low level details of upgrading your device.

You will start with an existing TI-RTOS example which does not have firmware upgrade capability. Working through this document, you will learn the steps necessary to add firmware upgrade capability. Using the techniques you have learned, you will be able to add the same capabilities to your own device application.

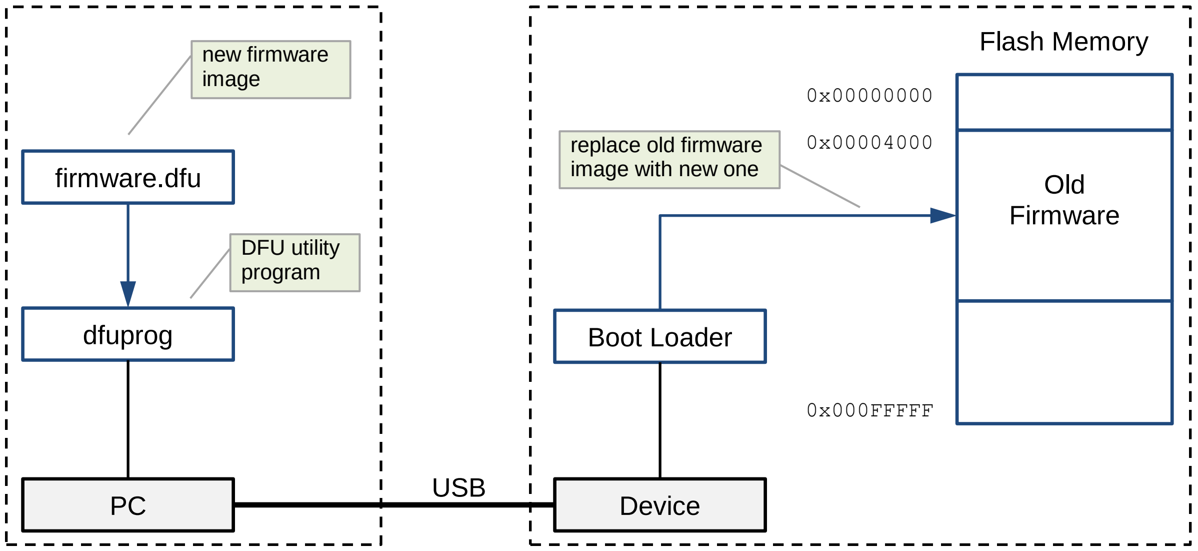
A key element to supporting firmware upgrade is a boot loader. The boot loader always runs first and determines if the device application should be started or if an upgrade has been requested. This example will show you how to add the boot loader to your device application. See Figure 1.

Figure 1 – Adding the boot loader



If upgrading the firmware, the boot loader communicates with the DFU utility running on the PC. The boot loader receives the new firmware and manages the details of writing it to flash. The DFU utility is a download available from Texas Instruments Incorporated. See Figure 2.

Figure 2 – Firmware Upgrade



You will also learn how to generate the new firmware image which you can then distribute to your end-users.

This example uses a push button to initiate the firmware upgrade. This has the advantage of not requiring any application changes because the boot loader enters upgrade mode before starting the application. However, some applications may not be able to use this technique. As an alternate method, this example also includes a project which uses a USB Composite interface which allows the device application to programmatically initiate the firmware upgrade.

For questions, comments, or technical support, please visit the TI-RTOS Forum:

http://e2e.ti.com/support/embedded/tirtos/

Setup: Install required products

Follow these instructions to satisfy the prerequisites for this example. You may substitute more recent product versions which are compatible with the versions listed below.

1. Install the following products in order to satisfy the prerequisites for this example.

   CCS 6.1.1
   TI-RTOS TivaC 2.14.04.31
   TivaWare C Series Windows Examples (SW-USB-win.2.1.1.71.msi)

   http://processors.wiki.ti.com/index.php/Download_CCS
   http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent
   http://software-dl.ti.com/tiva-c/SW-TM4C/latest/index_FDS.html

2. Setup the XDC path.

   In some cases, the required products will be included with your CCS installation. If so, simply select the products. Otherwise, download the products and install them yourself. Once installed, use the following instructions to inform CCS of the product locations. Repeat for each product.

   ```
   Window > Preferences
   Code Composer Studio > RTSC > Products
   Install New...
   Select the location of your installed product
   Restart CCS
   ```

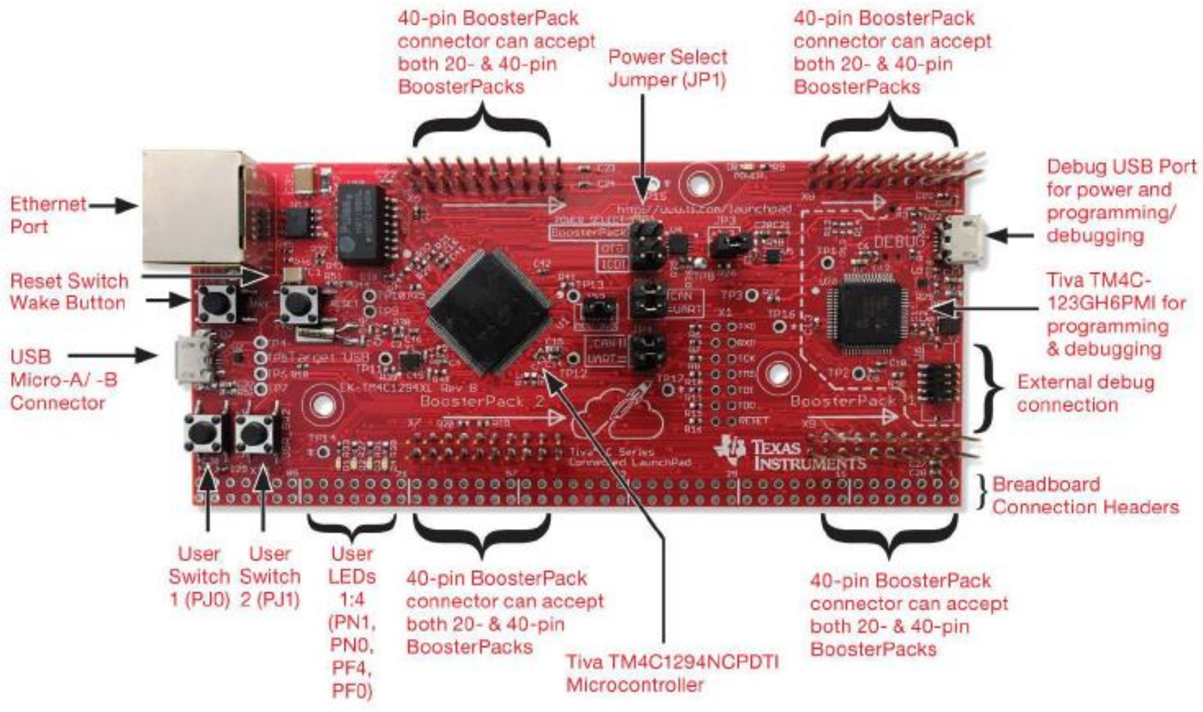
   Now you can specify your XDC path.

   ```
   Window > Preferences
   Code Composer Studio > RTSC
   XDCtools version > 3.31.1.33_core
   Products and Repositories > TI-RTOS for TivaC > 2.14.4.31 > Select
   ```

3. Connect your EK-TM4C1294XL Evaluation Kit to your computer.

   The evaluation kit has two USB receptacles. The USB Micro-A/B Connector (U7) is used by the application firmware to provide the USB functionality. This connector is also referred to as the application port. The Debug USB port (U22) is used by CCS for program download and target emulation control. Connect both of these USB ports to your computer. See Figure 3 – TM4C1294XL Evaluation Kit.

Figure 3 – TM4C1294XL Evaluation Kit


40-pin BoosterPack connector can accept both 20- & 40-pin BoosterPacks
Power Select Jumper (JP1)
40-pin BoosterPack connector can accept both 20- & 40-pin BoosterPacks
Ethernet Port
Debug USB Port for power and programming/ debugging
Reset Switch
Wake Button
Tiva TM4C-123GH6PMI for programming & debugging
USB Micro-A/ -B Connector
External debug connection
Breadboard Connection Headers
User Switch 1 (PJ0)
User Switch 2 (PJ1)
User LEDs 1:4 (PN1, PN0, PF4, PF0)
40-pin BoosterPack connector can accept both 20- & 40-pin BoosterPacks
Tiva TM4C1294NCPDTI Microcontroller
40-pin BoosterPack connector can accept both 20- & 40-pin BoosterPacks

The application port will not be recognized unless you have a valid application running on your target. The debug port will be recognized as a Stellaris In-Circuit Debug Interface. See the EK-TM4C1294XL User Guide for details on these ports.

Notes:

A. The instructions use a brief notation for listing the menu actions. For example, to issue a System Reset command in CCS you begin by selecting the Run menu. In this menu, you select the Reset submenu. Finally, you select the System Reset command from the submenu. These menu actions would be documented as follows:

```
Run > Reset > System Reset
```

The same format is used for clicking buttons and selecting options. Please note that RMB indicates you should press the right mouse button.

B. The instructions in this example assume that you are using the advanced settings view in your project properties dialog. To enable this view, click on the Show advanced settings link in the project properties dialog.

```
Select your project in the Project Explorer
Project > Properties
Click on Show advanced settings (lower left corner)
```

C. For clarity, the instructions in this example use the CCS menus to invoke actions. In many cases, the same action can be invoked using either a toolbar icon or a keyboard shortcut.

D. These instructions assume you have installed the example in the following folder. Please make the appropriate changes to reflect your actual install folder.

```
C:/TI/demo
```

E. Program addresses and sizes are dependent on tool chain and project settings. Your actual program addresses or sizes may differ slightly from those illustrated in this document.

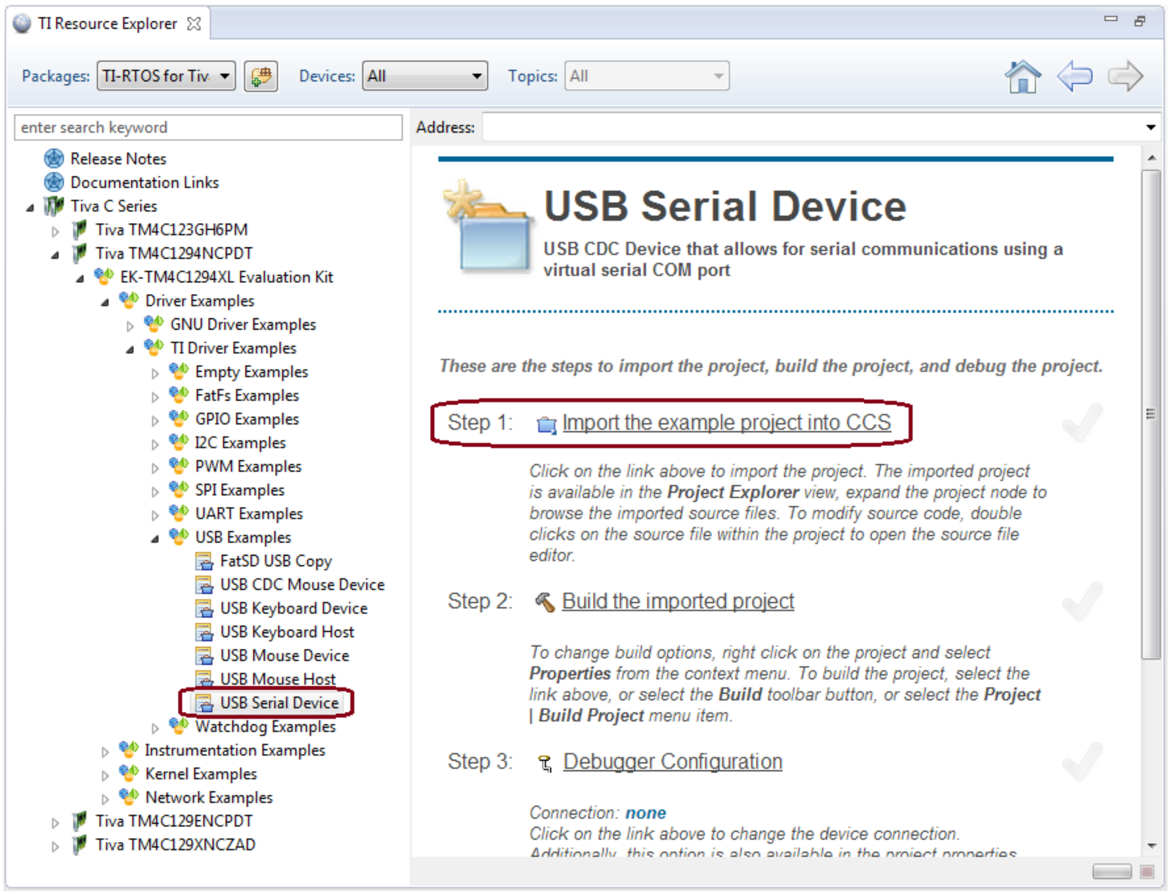
Part 1: Import USB Serial example

To get started, you will import the USB Serial Device example into your workspace, then build and run it on your evaluation kit. This will verify that you have a working setup.

1. Import the USB Serial Device example.

   In CCS, open the TI Resource Explorer. Navigate to your device and open the Driver Examples. Locate the USB Serial Device example and import it into your workspace (Step 1 only).

```
View > Resource Explorer
Packages > TI-RTOS for TivaC
Tiva C Series > Tiva TM4C1294NCPDT > EK-TM4C1294XL Evaluation Kit
Driver Examples > TI Driver Examples > USB Examples
USB Serial Device > Select
Click on Step 1: Import the example project into CCS
```



USB Serial Device

USB CDC Device that allows for serial communications using a virtual serial COM port

These are the steps to import the project, build the project, and debug the project.

Step 1: Import the example project into CCS

*Click on the link above to import the project. The imported project is available in the **Project Explorer** view, expand the project node to browse the imported source files. To modify source code, double clicks on the source file within the project to open the source file editor.*

Step 2: Build the imported project

*To change build options, right click on the project and select **Properties** from the context menu. To build the project, select the link above, or select the **Build** toolbar button, or select the **Project | Build Project** menu item.*

Step 3: Debugger Configuration

*Connection: **none***
Click on the link above to change the device connection.

This will make a complete copy of the example in your workspace. Modifying or deleting the project in your workspace will not change the original copy.

It’s a good idea to rename your new project, so that you may import the original project again if needed. This document assumes you have renamed the project as detailed next.

```
Select the project in the Project Explorer
File > Rename
Name: usbserial
```

Study the `Readme.txt` file in the project folder to learn more details about the example.

2. Create a target configuration for the application.

   Later in this example, the application requires a custom target configuration. To prepare for this, create a target configuration now which you will customize later.

   In order to manage your custom target configuration, you must first disable the automatic target configuration option in your project settings.

```
Project Explorer > usbserial > Select
Project > Properties
CCS General > Main (tab)
Manage the project's target-configuration automatically > Unselect
OK
```

   Create a target configuration for your project.

```
Project Explorer > usbserial > Select
File > New > Taget Configuration File
Name: usbserial.ccxml
Use shared location > Unselect
Workspace...
usbserial > Select
OK
Finish
```

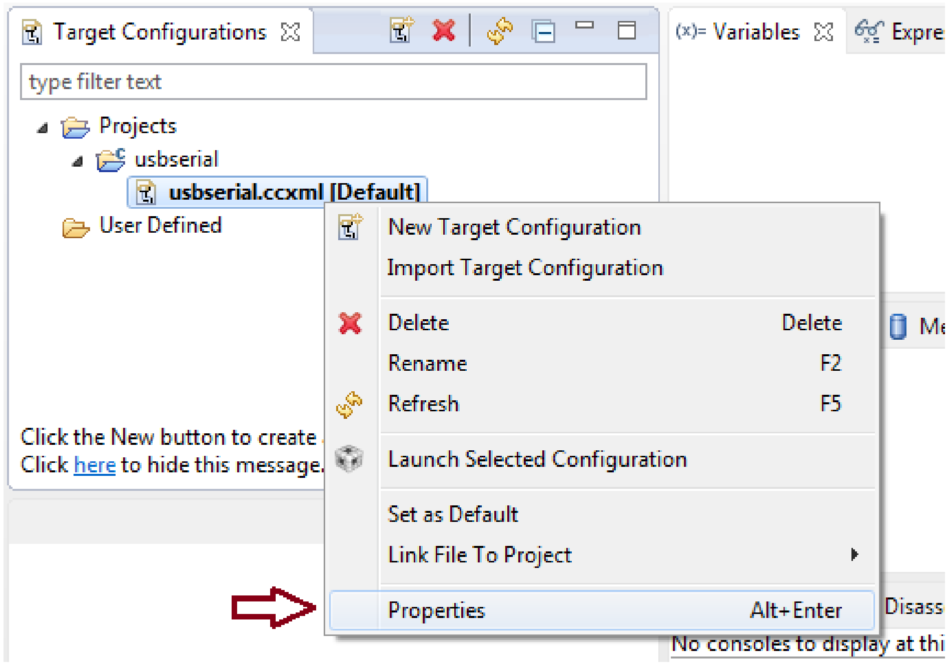
   This will create a new target configuration file (`usbserial.ccxml`) in your project folder and open the file in the edit window. Make the following edits and save your work.

```
Connection > Stellaris In-Circuit Debug Interface
Board or Device: TM4C1294
Tiva TM4C1294NCPDT > Select
Save
```

   At this point, you can close the target configuration file to free up your edit window.

   By default, when you download a program to the target, CCS will automatically run the target from the entry point (`_c_int00`) to the function `main()`. If the processor encounters an error before reaching `main()`, it may run continuously or get stuck in a reset loop. Disable the auto-run option to give yourself control of when the processor should run. This edit must be made in the properties dialog box for the target configuration. This dialog box is accessed from the Target Configuration view. Open this view if needed. Make the following selection to open the target configuration properties dialog box.

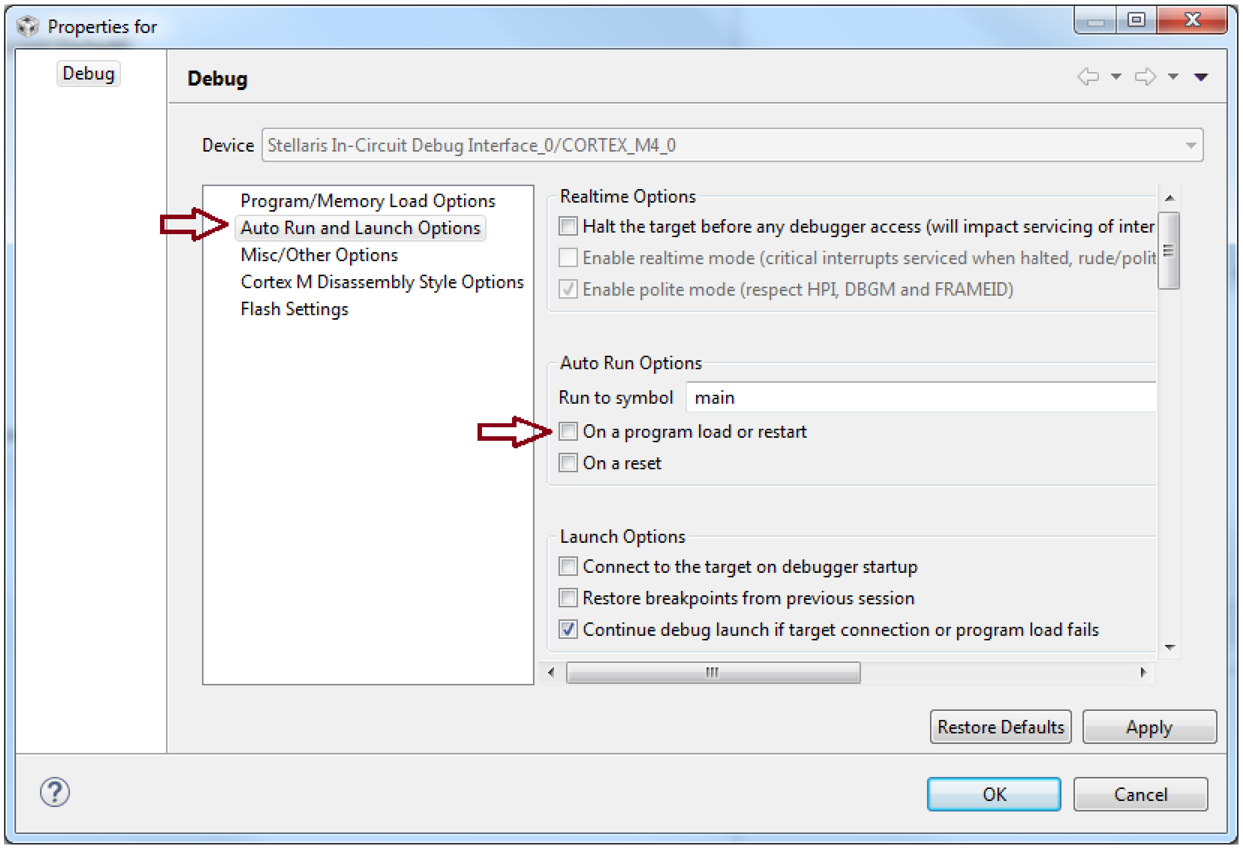
```
View > Target Configurations
Projects > usbserial > usbserial.ccxml > RMB > Properties
```



Click the New button to create
Click here to hide this message.

Make the following selection to disable the auto-run option.

```
Auto Run and Launch Options > Select
Auto Run Options > On a program load or restart > Unselect
OK
```


Properties for
Debug
Debug
Program/Memory Load Options
Auto Run and Launch Options
Misc/Other Options
Cortex M Disassembly Style Options
Flash Settings
Restore Defaults
Apply
OK
Cancel

Device Stellaris In-Circuit Debug Interface_0/CORTEX_M4_0

Realtime Options
- [ ] Halt the target before any debugger access (will impact servicing of inter
- [ ] Enable realtime mode (critical interrupts serviced when halted, rude/polit
- [x] Enable polite mode (respect HPI, DBGM and FRAMEID)

Auto Run Options
Run to symbol main
- [ ] On a program load or restart
- [ ] On a reset

Launch Options
- [ ] Connect to the target on debugger startup
- [ ] Restore breakpoints from previous session
- [x] Continue debug launch if target connection or program load fails

Finally, specify the flash crystal frequency to match the actual crystal on the evaluation kit.

```
View > Target Configurations
Projects > usbserial > usbserial.ccxml > RMB > Properties
Flash Settings
Crystal Frequency (MHz): 25
OK
```

3. Build the project.

   All build dependencies should be resolved by following the instructions in the setup section.

```
Project > Build Project
```

   When the build completes, you will have a usbserial.out file listed under your project's binaries section in the project explorer.

```
Project Explorer > usbserial
Binaries > usbserial.out
```

4. Load and run the program.

   Note: Don’t use the Run > Debug option because this will create a default launch configuration and not use the custom target configuration you created above.

   Use the Target Configurations view to launch your target configuration the first time.

```
View > Target Configurations
Projects > usbserial > usbserial.ccxml > RMB > Launch Selected Configuration
```

   Tip: Once you have launched your custom target configuration, it will be listed in the debug history menu (next to the bug icon on the Debug view toolbar). You may use this menu for subsequent launches.

   Now you must connect to the target and issue a CPU Reset.

```
Run > Connect Target
Run > Reset > Core Reset
```

   Download the application program to the target.

```
Run > Load > Load Program...
Browse project...
usbserial > Debug > usbserial.out
```

   Run the program.

```
Run > Resume
```

5. Install INF file.

   The first time you run this program on your evaluation kit, the USB device will not be recognized by your computer. You must install an INF file.

   Open the Windows Device Manager.

```
Windows > Control Panel
Hardware and Sound
Device Manager
View > Devices by type
```

Your device should be listed as a generic virtual COM port in the other devices section.

```
+-- Other devices
    |
    +-- Virtual COM Port
```

You need to install an INF file provided by the TivaWare product found in your TI-RTOS installation. Do the following in the Windows Device Manager.

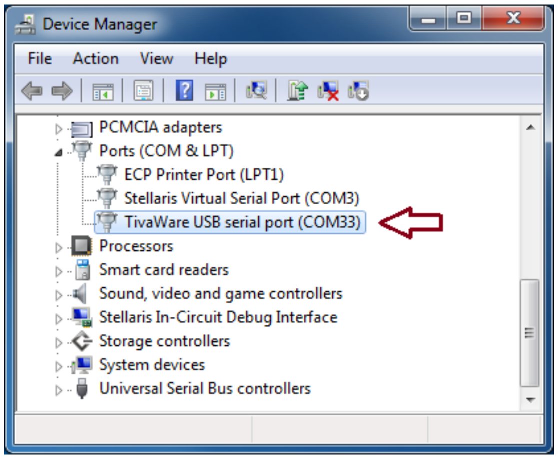
```
Virtual COM Port > RMB > Update Driver Software...
Browse my computer for driver software > Select
Browse...: <TI-RTOS Install>/products/TivaWare_C_Series-2.1.1.71b/windows_drivers
Include subfolders > Select
Next
```

The Windows Security dialog will inform you that it cannot verify the publisher of this driver. It is okay to proceed and install the driver anyway.

```
Windows Security Dialog
Install this driver software anyway > Select
```

After a short pause, Windows should have installed the INF file and reported that your device is a TivaWare USB serial port. When you close the dialog, the Windows Device Manager should refresh the device list.

Open the Ports (COM & LPT) entry. You should see a device similar to the following:


Device Manager
File
Action
View
Help
PCMCIA adapters
Ports (COM & LPT)
ECP Printer Port (LPT1)
Stellaris Virtual Serial Port (COM3)
TivaWare USB serial port (COM33)
Processors
Smart card readers
Sound, video and game controllers
Stellaris In-Circuit Debug Interface
Storage controllers
System devices
Universal Serial Bus controllers

Note the COM port number; you will need this when connecting your terminal program.

Use a terminal emulator to connect to your evaluation kit. CCS also provides a terminal emulator.

6. Use the CCS terminal window to connect to your device.

   Open the terminal window.

```
View Other > Terminal > Terminal
```

Configure the serial connection properties before connecting to your device.

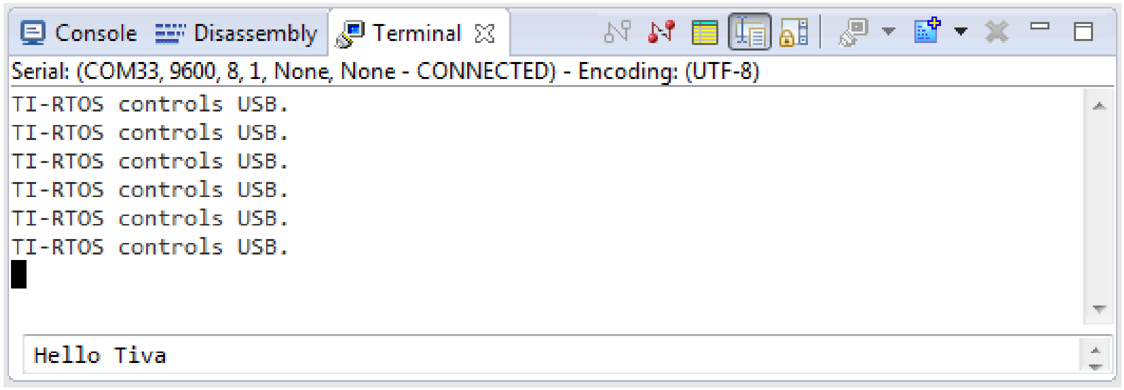
```
Terminal (toolbar) > Settings
Encoding > UTF-8
Connection Type > Serial
Port > COM33
```

Click the connect icon in the toolbar to connect the terminal window to your device. Note that only one terminal program can be used at a time. If you have another terminal program connected to your device you must first disconnect it.

```
Terminal (toolbar) > Connect
```

You can also send text characters to your device. Open the command input field in your terminal window and enter some text. The following step will show you how to see this text on your device.

```
Terminal (toolbar) > Toggle Command Input Field > Select
Enter: Hello Tiva (press return to send)
```



```
Serial: (COM33, 9600, 8, 1, None, None - CONNECTED) - Encoding: (UTF-8)
TI-RTOS controls USB.
TI-RTOS controls USB.
TI-RTOS controls USB.
TI-RTOS controls USB.
TI-RTOS controls USB.
TI-RTOS controls USB.

Hello Tiva
```

7. Inspect the data received by the device.

When you send text to the device, the data is stored in a buffer. Use the RTOS Object view to inspect the data on the target. To use ROV, you must first pause the processor.

```
Run > Suspend
Tools > RTOS Object View (ROV)
usbserial.out > Viewable Modules > SysMin > Select
Output Buffer (tab) > Select
```

You should see your text along with some additional annotation.

8. Shutdown the example.

When you are finished, it is a good idea to take your device off the USB bus. To do this, halt the processor and issue a system reset. This will remove the device from the USB bus without physically disconnecting the USB cable.

```
Run > Suspend
Run > Reset > System Reset
```

The device should no longer be visible in the Windows Device Manager.

Part 2: Build the TivaWare boot loader

In order to add USB Device Firmware Upgrade (DFU) support to your application, you must have a boot loader on your device. This section shows how to create a CCS project to build the TivaWare boot loader.

You have two choices on how to work through this section: either import the completed boot loader project, or build up the project from scratch. To import the project, complete steps 1 and 2, then skip to step 10. Otherwise, start with step 3 to build up the project from scratch. Either way, you must complete step 10 and all remaining steps.

1. Import the boot loader project.

   The boot loader project is located in the `ccs` folder of this example. It is named `bootloaderusb_completed`. After you import this project into your workspace, rename it to bootloaderusb.

   ```
   Project > Import CCS Projects...
   Select search-directory: ek_tm4c1294xl_usb_serial_dfu/ccs
   bootloaderusb_completed > Select
   Copy projects into workspace > Select
   Finish
   ```

   Rename the bootloader project.

   ```
   Project Explorer > bootloaderusb_completed > Select
   File > Rename
   New Name: bootloaderusb
   ```

   Rename the target configuration file.

   ```
   Project Explorer > bootloaderusb > bootloaderusb_completed.ccxml > Select
   File > Rename
   New Name: bootloaderusb.ccxml
   ```

2. Build the boot loader.

   At this point, you should be able to build your project and generate an executable file.

   ```
   Project > Build Project
   ```

   When the build completes, you should have the file `bootloaderusb.out` listed under your project's binaries section in the project explorer.

   ```
   Project Explorer > bootloaderusb > Binaries
   bootloaderusb.out
   ```

   Now skip to Step 10 Customize the target configuration.

3. Create a new CCS project for the boot loader.

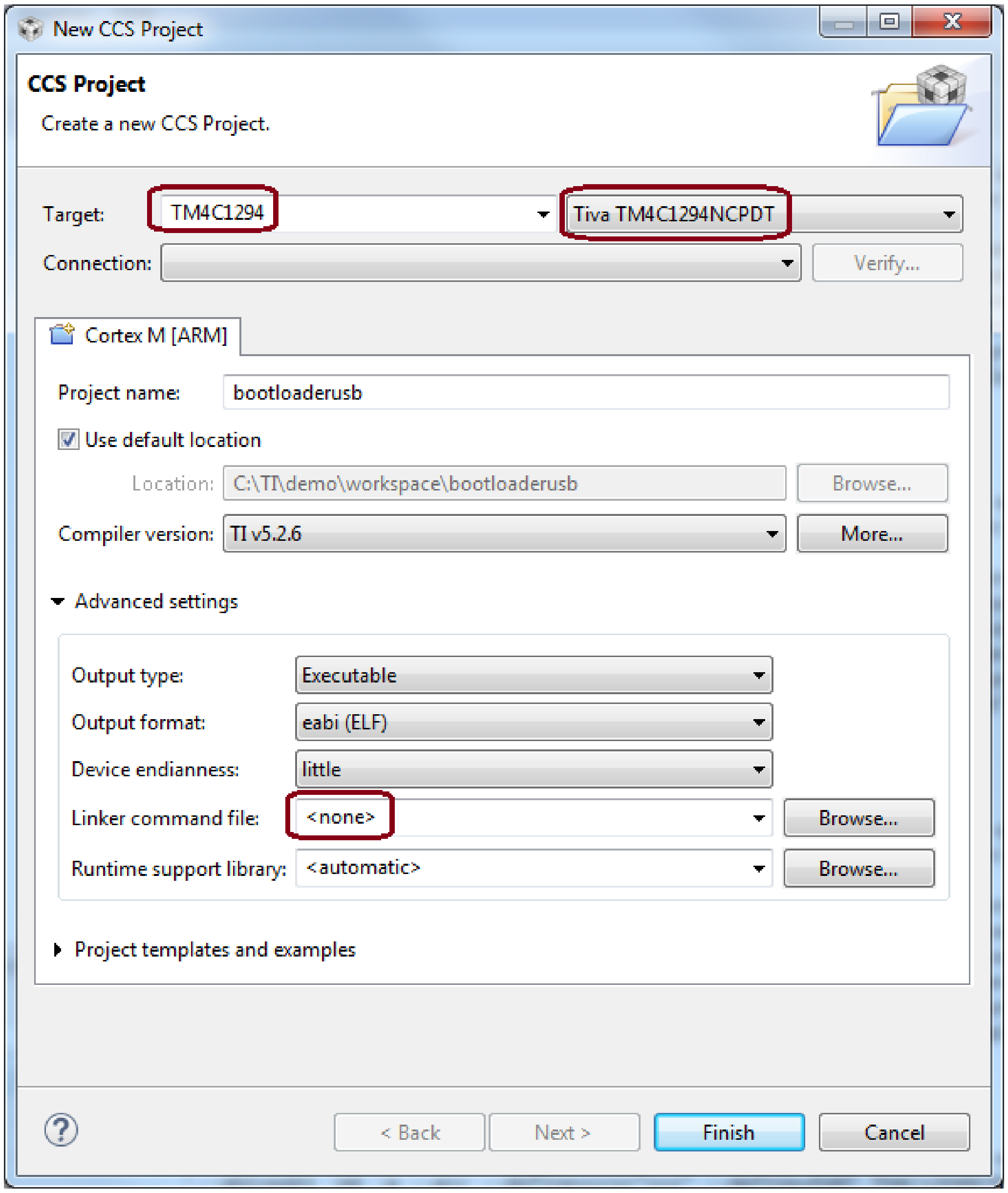
   This project will build a boot loader specific to your platform and application. Keep this in mind when choosing a project name. Note you will not choose a debug connection, leave it blank. The debug configuration will be created in the next step.

   The project wizard has two pulldown menus for selecting the target device: a filter menu and a device menu. The filter menu (on the left) is used to limit the number of items displayed in the device menu (on the right). Enter the string "TM4C1294" in the filter menu; this will make the device menu much shorter.

```
Project > New CCS Project...
Target Filter: TM4C1294
Device Menu > Tiva TM4C1294NCPDT
Connection > <blank>
Project Name: bootloaderusb
Use default location > Select

Advanced settings > Open
Linker command file > <none>

Project templates and examples > Open
Empty Projects > Empty Project > Select
Finish
```



Target: TM4C1294 Tiva TM4C1294NCPDT
Connection:
Project name: bootloaderusb
Use default location ✓
Location: C:\TI\demo\workspace\bootloaderusb
Compiler version: TI v5.2.6
Output type: Executable
Output format: eabi (ELF)
Device endianness: little
Linker command file: <none>
Runtime support library: <automatic>

4. Create a target configuration for the boot loader.

   The boot loader requires a custom target configuration; you cannot let CCS manage it for you. In particular, after downloading the boot loader to the target, CCS must issue a Core Reset. If this is not done, the boot loader will not run correctly. In addition, the boot loader has no function called `main()`. You should disable the auto-run to main option.

   Note: There is a subtle difference in how the processor behaves on a power on system reset when compared to a CCS load program command. On system reset, the processor reads the value at address `0x00000004` (reset vector) and loads that value into the PC register. However, when CCS loads a program, it will remember the address of the program's entry point (`_c_int00`) and loads that value into the PC register. In most cases, the reset vector and the entry point are the same. But the boot loader is different. The reset vector contains the address of a function (`ProcessorInit`) which must run before jumping to the entry point. Therefore, it is important to always issue a Core Reset before running the boot loader program. You can issue the Core Reset manually from the menu. In addition, the following target configuration settings will automatically issue a Core Reset whenever you load a new program to the target.

   In order to manage your custom target configuration, you must first disable the automatic target configuration option in your project settings.

```
Project Explorer > bootloaderusb > Select
Project > Properties
CCS General > Main (tab)
Manage the project's target-configuration automatically > Unselect
OK
```

   Create a target configuration for you project.

```
Project Explorer > bootloaderusb > Select
File > New > Target Configuration File
Name: bootloaderusb.ccxml
Use shared location > Unselect
Workspace
bootloaderusb > Select
OK
Finish
```

   This will create a new target configuration file (`bootloaderusb.ccxml`) in your project folder and open the file in the edit window. Make the following edits and save your work.

```
Connection > Stellaris In-Circuit Debug Interface
Board or Device: TM4C1294
Tiva TM4C1294NCPDT > Select
Save
```

   At this point, you can close the target configuration file to free up your edit window.

5. Add the boot loader source files to your project.

   The boot loader source files are distributed with this example. However, before importing the source files, you must delete the existing startup file from your project. You will use the startup file provided by the boot loader.

```
Project Explorer > bootloaderusb
tm4c1294ncpdt_startup_ccs.c > Select
Edit > Delete
```

The boot loader source files must be in a folder called `boot_loader` within your project. Create this folder in your project.

```
Project Explorer > bootloaderusb
File > New > Folder
Enter or select the parent folder: bootloaderusb
Folder name: boot_loader
Advanced > Use default location > Select
Finish
```

Add the boot loader source files to your project by copying them from the `boot_loader` folder in this example. The boot loader supports several different development environments and update transports. You don't need all of them for this example. Since you are using CCS, and are building for USB support, copy only the following files into your project.

Make sure you select the `boot_loader` folder you just created. This will be the destination folder for the boot loader source files.

```
Project Explorer > bootloaderusb > boot_loader > Select
Project > Add Files...
Navigate to the boot loader source folder in the example
ek_tm4c1294xl_usb_serial_dfu/boot_loader
```

Select the following files:

```
bl_check.c
bl_check.h
bl_commands.h
bl_config.h.templ
bl_crc32.c
bl_crc32.h
bl_crystal.h
bl_decrypt.c
bl_decrypt.h
bl_flash.c
bl_flash.h
bl_hooks.h
bl_i2c.h
bl_link_ccs.cmd
bl_main.c
bl_packet.h
bl_ssi.h
bl_startup_ccs.s
bl_uart.h
bl_usb.c
bl_usbfuncs.c
bl_usbfuncs.h
readme.txt
usbdfu.h

Click Open
Copy files > Select
OK
```

The boot loader configuration is controlled by the file `bl_config.h`. This file ships under a different name; you must rename it.

```
Project Explorer > bootloaderusb > boot_loader > bl_config.h.tmpl > Select
File > Rename
New name: bl_config.h
OK
```

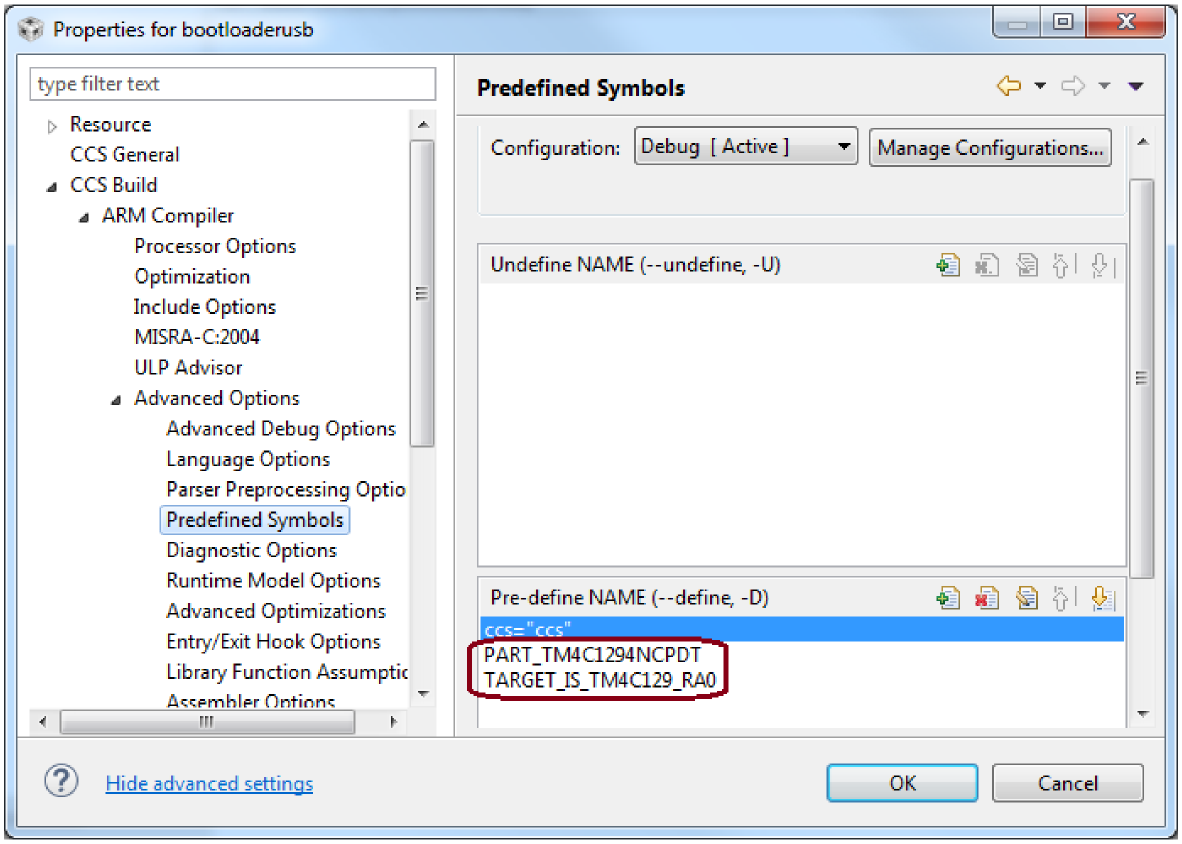
6. Add compiler build options to your CCS project.

   Note these build options must be set for each build configuration (e.g. Debug, Release).

   Add the following symbols to the compiler predefined symbols list.

```
Project Explorer > bootloaderusb
Project > Properies
CCS Build > ARM Compiler > Advanced Options > Predefined Symbols

Add the following two symbols:
PART_TM4C1294NCPDT
TARGET_IS_TM4C129_RA0
```



Configuration: Debug [Active]

```
ccs="ccs"
PART_TM4C1294NCPDT
TARGET_IS_TM4C129_RA0
```

Add the following include paths to the compiler include options.

```
Project Explorer > bootloaderusb
Project > Properies
CCS Build > ARM Compiler > Include Options

"${CG_TOOLS_ROOT}/include"
"${COM_TI_RTSC_TIRTOSTIVAC_INSTALL_DIR}/products/TivaWare_C_Series-2.1.1.71b"
"${workspace_loc:/${ProjName}}"
```

Set the following advanced compiler options.

```
Project Explorer > bootloaderusb
Project > Properies
CCS Build > ARM Compiler > Advanced Options > Runtime Model Options
Place each function in a separate subsection > On
CCS Build > ARM Compiler > Advanced Options > Assembler Options
Use unified assembler language > Select
```

Set the following linker options.

```
Project Explorer > bootloaderusb
Project > Properies
CCS Build > ARM Linker > File Search Path
Reread libraries; resolve backward references > Unselect
CCS Build > ARM Linker > Advanced Options > Runtime Environment
Initialization model > Link using RAM autoinitialization model
```

7. Configure the boot loader build options.

   Use the following configuration values for this example.

   Edit `boot_loader/bl_config.h`

   Use the following values for this example. When integrating the boot loader with your device application, you may need to adjust some of these. The `APP_START_ADDRESS` is the first flash block boundary after the end of the boot loader.

```
#define CRYSTAL_FREQ                    25000000
#define APP_START_ADDRESS               0x00004000
#define VTABLE_START_ADDRESS            0x00004000
#define FLASH_PAGE_SIZE                 0x4000
#define STACK_SIZE                      128
```

   The following USB Vendor ID has been reserved by Texas Instruments Incorporated. You may use this value for your initial development, but you must acquire your own Vendor ID from the USB-IF. Also, please note that the Product ID must be different from that used by your device application.

```
#define USB_ENABLE_UPDATE
#define USB_VENDOR_ID                   0x1cbe
#define USB_PRODUCT_ID                  0x00ff
#define USB_DEVICE_ID                   0x0001
```

   The following values enable the USB controller on the EK-TM4C1294XL Evaluation Kit. You will need to update these values if using another evaluation kit or your own custom board.

```
#define USB_MAX_POWER               150
#define USB_BUS_POWERED             1
#define USB_VBUS_CONFIG
#define USB_VBUS_PERIPH             SYSCTL_RCGCGPIO_R1
#define USB_VBUS_PORT               GPIO_PORTB_BASE
#define USB_VBUS_PIN                1
#define USB_ID_CONFIG
#define USB_ID_PERIPH               SYSCTL_RCGCGPIO_R1
#define USB_ID_PORT                 GPIO_PORTB_BASE
#define USB_ID_PIN                  0
#define USB_DP_CONFIG
#define USB_DP_PERIPH               SYSCTL_RCGCGPIO_R10
#define USB_DP_PORT                 GPIO_PORTL_BASE
#define USB_DP_PIN                  6
#define USB_DM_CONFIG
#define USB_DM_PERIPH               SYSCTL_RCGCGPIO_R10
#define USB_DM_PORT                 GPIO_PORTL_BAS E
#define USB_DM_PIN                  7
```

These config params enable the pin-based update mode. Configure push button User 1 for a press action. Port J, Pin 0.

```
#define ENABLE_UPDATE_CHECK
#define FORCED_UPDATE_PERIPH        SYSCTL_RCGCGPIO_R8
#define FORCED_UPDATE_PORT          GPIO_PORTJ_BASE
#define FORCED_UPDATE_PIN           0
#define FORCED_UPDATE_POLARITY      0
#define FORCED_UPDATE_WPU
```

8. Build the boot loader.

   At this point, you should be able to build your project and generate an executable file.

   ```
   Project > Build Project
   ```

   When the build completes, you should have the file `bootloaderusb.out` listed under your project's binaries section in the project explorer.

   ```
   Project Explorer > bootloaderusb > Binaries
   bootloaderusb.out
   ```

9. Create a target configuration for the boot loader.

   The boot loader requires a custom target configuration; you cannot let CCS manage it for you. In particular, after downloading the boot loader to the target, CCS must issue a Core Reset. If this is not done, the boot loader will not run correctly. In addition, the boot loader has no function called `main()`. You should disable the auto-run to main option.

   In order to manage your custom target configuration, you must first disable the automatic target configuration option in your project settings.

   ```
   Project Explorer > bootloaderusb > Select
   Project > Properties
   CCS General > Main (tab)
   Manage the project's target-configuration automatically > Unselect
   OK
   ```

   Create a target configuration for you project.

```
Project Explorer > bootloaderusb > Select
File > New > Target Configuration File
Name: bootloaderusb.ccxml
Use shared location > Unselect
Workspace
bootloaderusb > Select
OK
Finish
```

This will create a new target configuration file (`bootloaderusb.ccxml`) in your project folder and open the file in the edit window. Make the following edits and save your work.

```
Connection > Stellaris In-Circuit Debug Interface
Board or Device: TM4C1294
Tiva TM4C1294NCPDT > Select
Save
```

At this point, you can close the target configuration file to free up your edit window.

10. Customize the target configuration.

    Enable the reset target on program load option and turn off the auto-run to main option in your target configuration. These edits must be made from the Target Configuration view. Open this view if needed.

```
View > Target Configurations
Projects > bootloaderusb > bootloaderusb.ccxml > RMB > Properties
Program/Memory Load Options > Select
Reset the target on a program load or restart > Select
Apply

Auto Run and Launch Options > Select
Auto Run Options > On a program load or restart > Unselect
OK
```

11. Load and run the boot loader.

    A custom target configuration must always be launched explicitly. Use the Debug Configuration dialog for the first launch.

```
Run > Debug Configurations
Code Composer Studio > bootloaderusb.ccxml > Select
Debug
```

Launching a target configuration starts a new debug session. Use the CCS Debug perspective to interact with your debug session (it should have been opened when launching your target configuration).

Tip: Once you have launched your custom target configuration, it will be listed in the debug history menu (next to the bug icon on the Debug view toolbar). You may use this menu for subsequent launches.

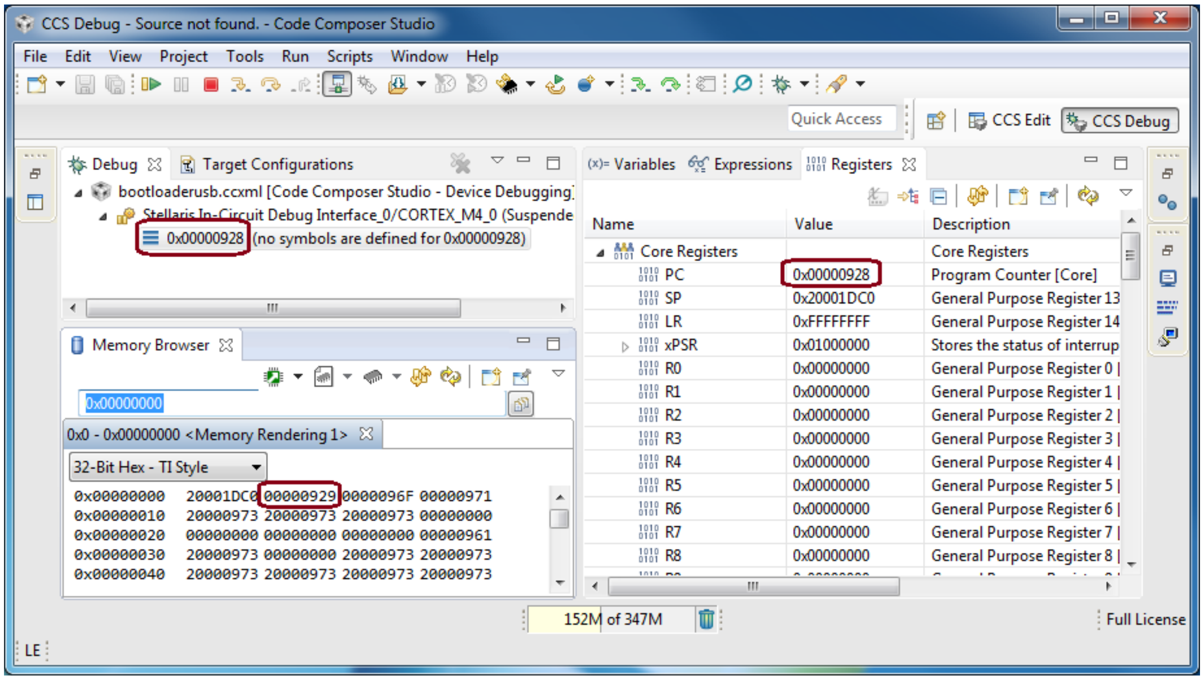
Now you must connect to the target and issue a Core Reset.

```
Run > Connect Target
Run > Reset > Core Reset
```

Download the boot loader program to the target.

```
Run > Load > Load Program...
Browse project...
bootloaderusb > Debug > bootloaderusb.out
```

After the download completes, the PC should be at the reset vector, not `_c_int00`. If this is not the case, double-check the launch configuration settings. The reset vector is defined at address `0x00000004` (the second entry in the vector table). Note that bit-0 of the reset vector is set to 1, but the PC value is always even. Use a memory browser to confirm this address.



```
bootloaderusb.ccxml [Code Composer Studio - Device Debugging]
  Stellaris In-Circuit Debug Interface_0/CORTEX_M4_0 (Suspende
    0x00000928 (no symbols are defined for 0x00000928)
```

```
0x00000000
0x0 - 0x00000000 <Memory Rendering 1>
0x00000000  20001DC0 00000929 0000096F 00000971
0x00000010  20000973 20000973 20000973 00000000
0x00000020  00000000 00000000 00000000 00000961
0x00000030  20000973 00000000 20000973 20000973
0x00000040  20000973 20000973 20000973 20000973
```

Name	Value	Description
Core Registers		Core Registers
PC	0x00000928	Program Counter [Core]
SP	0x20001DC0	General Purpose Register 13
LR	0xFFFFFFFF	General Purpose Register 14
xPSR	0x01000000	Stores the status of interrup
R0	0x00000000	General Purpose Register 0 |
R1	0x00000000	General Purpose Register 1 |
R2	0x00000000	General Purpose Register 2 |
R3	0x00000000	General Purpose Register 3 |
R4	0x00000000	General Purpose Register 4 |
R5	0x00000000	General Purpose Register 5 |
R6	0x00000000	General Purpose Register 6 |
R7	0x00000000	General Purpose Register 7 |
R8	0x00000000	General Purpose Register 8 |

When you download the boot loader to your evaluation kit, the launch configuration will erase all of flash before it writes the boot loader into flash. As a result, when the boot loader starts, it will automatically enter upgrade mode because there is no valid firmware in flash. Run the program now.

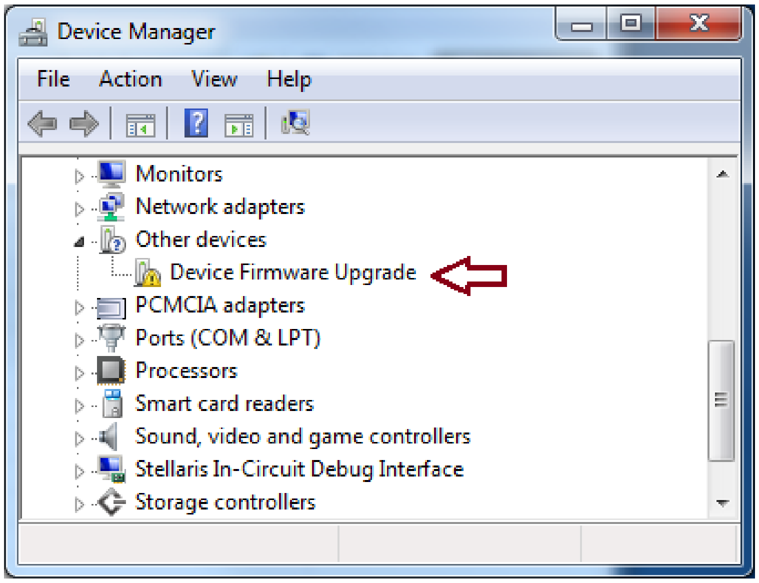
```
Run > Resume
```

The first time you run this program on your evaluation kit, the USB device will not be recognized by your computer. You must install an INF file.

Open the Windows Device Manager.

```
Windows > Control Panel
Hardware and Sound
Device Manager
View > Devices by type
```

Your device should be listed in the other devices section.



You need to install an INF file provided by the TivaWare product found in your TI-RTOS installation. Do the following in the Windows Device Manager.

```
Other devices > Device Firmware Upgrade > RMB > Update Driver Software...
Browse my computer for driver software > Select
Browse...: <TI-RTOS Install>/products/TivaWare_C_Series-2.1.1.71b/windows_drivers
Include subfolders > Select
Next
```

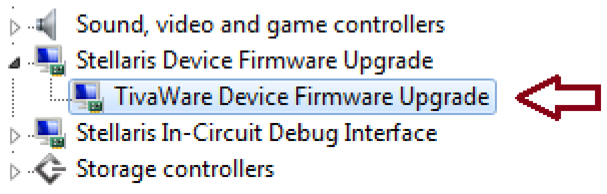
The Windows Security dialog will inform you that it cannot verify the publisher of this driver. It is okay to proceed and install the driver anyway.

```
Windows Security Dialog
Install this driver software anyway > Select
```

After a short pause, Windows should have installed the INF file and reported that your device is a TivaWare Device Firmware Upgrade. When you close the dialog, the Windows Device Manager should refresh the device list.

At this point, the boot loader will have entered USB DFU mode and enumerated itself as a Stellaris Device Firmware Upgrade device.

You should see the following device in the Windows Device Manager.


Sound, video and game controllers
Stellaris Device Firmware Upgrade
TivaWare Device Firmware Upgrade
Stellaris In-Circuit Debug Interface
Storage controllers

12. Remove the device from the bus.

Before moving on to the next part, it is a good idea to remove the device from the USB bus. To do this, halt the processor and issue a system reset. This will remove the device from the USB bus without physically disconnecting the USB cable.

```
Run > Suspend
Run > Reset > System Reset
```

The device should no longer be visible in the Windows Device Manager.

Part 3: Combine boot loader with application

By default, both the boot loader and your application are linked to a start address of `0x00000000`, the beginning of flash memory. In order to place both in flash at the same time, you must move the start address of your application past the end of the boot loader. In addition, the start address must be aligned on a flash block boundary (`0x4000` for the TM4C1294XL).

In this section, you will modify the application memory map as described above, download both the boot loader and the application to the target and boot the device into device runtime mode.

1. Determine how much flash memory the boot loader requires.

   Inspect the memory map file for your boot loader. It is located in the debug folder of the boot loader project.

   ```
   Project Explorer > bootloaderusb > Debug
   bootloaderusb.map > double-click
   ```

   The map file is loaded into the text editor window. At the top of the file, you will find the memory configuration. Note the amount of flash used by the boot loader.

   ```
   MEMORY CONFIGURATION
            name            origin    length      used     unused
   ----------------------  --------  ---------  --------  --------
     FLASH                 00000000   00100000  00001710  000fe8f0
     SRAM                  20000000   00040000  00001da8  0003e258
   ```

   ```
   flash used = 0x1710
   ```

   Round this value up to the next flash block boundary. This will be the start address for the application.

   ```
   application start address = 0x00004000
   ```

   The start address of the application effectively partitions the flash memory into two segments: one for the loader and one for the application. It is a good idea to edit the memory map for both the boot loader and the application to reflect this partitioning. This way, if either program exceeds their respective memory segments, you will get a link error, which is much easier to fix than debugging a runtime failure.

   ```
   Unified Flash Memory Map
   Address     Size    Comment
   --------------------------------------------------
   00000000     4000   boot loader
   00004000    FC000   application
   ```

   The SRAM memory is not partitioned. When the respective executable is running, each one has full access to the entire SRAM. You do not need to make any changes to the memory map regarding SRAM.

2. Modify the memory map for the application.

   Edit the application linker command file. In the application project, open the linker command file (.cmd) in the text editor.

```
Project Explorer > usbserial
EK_TM4C1294XL.cmd > double-click
```

Edit the memory directive to reflect the flash memory map above.

```
MEMORY
{
    FLASH (RX) : origin = 0x00004000, length = 0x000FC000
    SRAM (RWX) : origin = 0x20000000, length = 0x00040000
}
```

You must also place the vector table at the application start address. The default configuration is to place the vector table at address zero. The vector table address is specified in the application configuration script.

Use the XDCscript editor to add the following code to set the new vector table address.

```
Project Explorer > usbserial
usbserialdevice.cfg > RMB > Open With > XDCscript Editor
```

Scroll to the bottom of the script and add the following statements.

```
/* place vector table at application start address */
var ti_sysbios_family_arm_m3_Hwi = xdc.useModule('ti.sysbios.family.arm.m3.Hwi');
ti_sysbios_family_arm_m3_Hwi.resetVectorAddress = 0x00004000;
```

Build the application.

```
Project > Build Project
```

Open the generated map file. Inspect the memory configuration. Verify it is using the new memory map.

```
Project Explorer > usbserial > Debug
usbserial.map > double-click
```

```
MEMORY CONFIGURATION
         name            origin    length      used     unused
----------------------  --------  ---------  --------  --------
  FLASH                 00004000   000fc000  00008a9a  000f3566
  SRAM                  20000000   00040000  00001cb9  0003e347
```

3. Modify the memory map for the boot loader.

Edit the boot loader linker command file. In the boot loader project, open the linker command file (.cmd) in the text editor.

```
Project Explorer > bootloaderusb > boot_loader
bl_link_ccs.cmd > double-click
```

Edit the memory directive to reflect the flash memory map above.

```
MEMORY
{
    FLASH (RX) : origin = 0x00000000, length = 0x00004000
    SRAM (RWX) : origin = 0x20000000, length = 0x00040000
}
```

Build the boot loader.

```
Project > Build Project
```

Open the generated map file. Inspect the memory configuration. Verify it is using the new memory map.

```
Project Explorer > bootloaderusb > Debug
bootloaderusb.map > double-click
```

```
MEMORY CONFIGURATION

         name            origin    length      used     unused   
----------------------  --------  ---------  --------  --------
  FLASH                 00000000   00004000  00001710  000028f0
  SRAM                  20000000   00040000  00001da8  0003e258
```

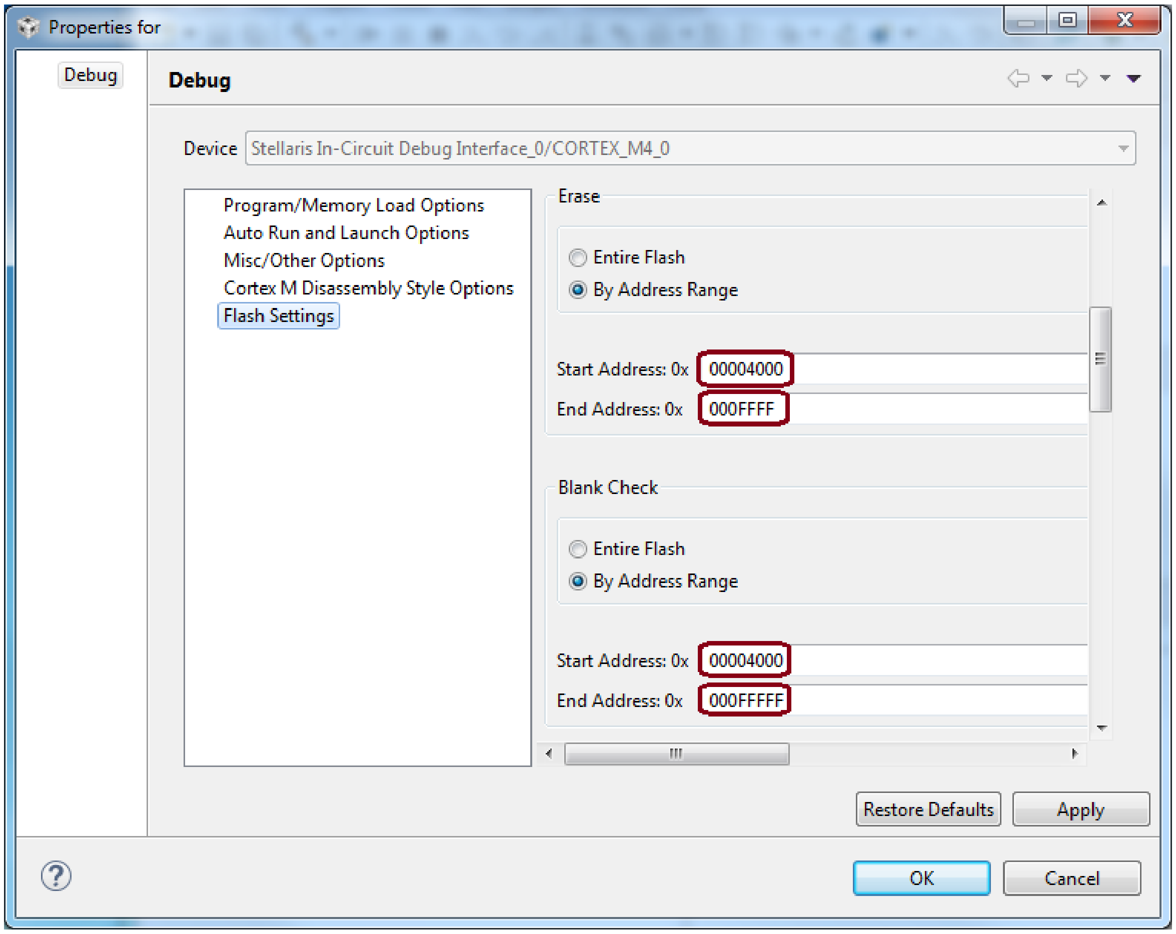
4. Customize target configuration for the application.

   When downloading an executable to the target, the default target configuration will erase all of flash memory. In order to download the application without erasing the boot loader already in flash, you must specify the flash memory erase behavior.

   Edit the flash settings in the application target configuration. Specify the address range allocated to your application. This will preserve the contents of flash memory occupied by the boot loader.

   These edits must be made from the Target Configuration view. Open this view if needed.

```
View > Target Configurations
Projects > usbserial > usbserial.ccxml > RMB > Properties
Flash Settings > Select
Erase > By Address Range > Select
Start Address: 00004000 (omit the leading 0x)
End Address: 000FFFFF
Blank Check > By Address Range > Select
Start Address: 00004000
End Address: 000FFFFF
OK
```


Properties for
Debug
Debug
Program/Memory Load Options
Auto Run and Launch Options
Misc/Other Options
Cortex M Disassembly Style Options
Flash Settings
Restore Defaults
Apply
OK
Cancel

Device Stellaris In-Circuit Debug Interface_0/CORTEX_M4_0

Erase

- ( ) Entire Flash
- (●) By Address Range

Start Address: 0x 00004000

End Address: 0x 000FFFF

Blank Check

- ( ) Entire Flash
- (●) By Address Range

Start Address: 0x 00004000

End Address: 0x 000FFFFF

5. Download both the boot loader and application to the target.

   When downloading either the boot loader or the application to the target, you must use the respective target configurations created above. Remember to shut down the debug session and launch the project's target configuration when switching projects.

   You must first download the boot loader image because that project's target configuration will erase the entire flash memory. After that, you will restart your debug session using the application project's target configuration in order to also download the application image without erasing the boot loader.

   Shut down any active debug session.

```
Run > Disconnect Target
Run > Terminate
Click Remove All Terminated Launches (icon on the debug toolbar)
```

   Launch the boot loader target configuration.

```
Run > Debug History > bootloaderusb.ccxml
```

   Connect to the target and reset the processor.

```
Run > Connect Target
Run > Reset > Core Reset
```

Download the boot loader.

```
Run > Load > Load Program
Browse project...
bootloaderusb > Debug > bootloaderusb.out
```

Now terminate your debug session.

```
Run > Disconnect Target
Run > Terminate
Click Remove All Terminated Launches
```

Launch the application target configuration.

```
Run > Debug History > usbserial.ccxml
```

Connect to the target and reset the processor.

```
Run > Connect Target
Run > Reset > Core Reset
```

Download the application.

```
Run > Load > Load Program
Browse project...
usbserial > Debug > usbserial.out
```

Reset the processor and run it.

```
Run > Reset > Core Reset
Run > Resume
```

You should now see your USB Serial device listed in the Windows Device Manager.

6. Test the target in USB device mode.

At this point, your target is ready for USB device runtime mode. Terminate your CCS debug session and disconnect both USB cables from the evaluation kit.

Move the power selector jumper (JP1) from the ICDI position to the OTG position. The evaluation kit will now be powered from the USB application receptacle.

Reconnect the cable to the USB application receptacle (don't connect the USB Debug port). The target should power up and connect to the USB bus in device runtime mode. Open the Windows Device Manager. You should see your device listed as a USB serial port.

```
+-- Ports (COM & LPT)
     |
     +-- TivaWare USB serial port (COM33)
```

You can use the Terminal view in CCS to connect to the COM port and observe that it is actually running.

You have successfully made a USB device with a resident boot loader which is used to bootstrap your application.

Part 4: Firmware upgrade image

In this part, you will add a counter to the transmit message. In addition, you will add a post build command to the CCS application project to generate a firmware upgrade image.

1. Copy the existing application project.

   Before making any changes to the existing project, let's make a copy of it. This makes it easy to go back to the original application if needed. First, clean the application project.

   ```
   Project Explorer > usbserial > RMB > Clean Project
   ```

   Now copy and paste the project.

   ```
   Project Explorer > usbserial > Select
   Edit > Copy
   Edit > Paste

   Project name: usbserial-01
   Use default location > Select
   OK
   ```

2. Add a tick count to the transmit message.

   Open the usbserialdevice.c source file in the editor.

   ```
   Project Explorer > usbserial-01 > usbserialdevice.c > double-click
   ```

   Add a symbol to define the size of your character buffer. We will use this buffer to prepend a tick count to the original text message. Note the reference to the symbol `text`.

   ```
   #define CBUF_SIZE (16 + sizeof(text))
   ```

   In the transmit function, use the System_snprintf() function to prepend the current tick count to the text message. Modify the function as indicated below. The changes are highlighted.

   ```
   Void transmitFxn(UArg arg0, UArg arg1)
   {
       unsigned int count = 0;
       unsigned char cbuf[CBUF_SIZE];
       int nb;

       while (true) {

           /* Block while the device is NOT connected to the USB */
           USBCDCD_waitForConnect(BIOS_WAIT_FOREVER);

           nb = System_snprintf((Char *)cbuf, CBUF_SIZE, "[%u] %s", ++count, text);
           USBCDCD_sendData(cbuf, nb+1, BIOS_WAIT_FOREVER);
           GPIO_toggle(Board_LED0);

           /* Send data periodically */
           Task_sleep(2000);
       }
   }
   ```

   Build your project.

3. Generate a binary image file.

   Add the following post build command to generate a binary image file from the linker generated executable file.

```
Project Explorer > usbserial-01 > Select
Project Properties
CCS Build > Steps (tab)
Post-build steps
```

   Add the following command, all on one line with a space separating each of the quoted strings below.

```
"${CCS_INSTALL_ROOT}/utils/tiobj2bin/tiobj2bin"
"${BuildArtifactFileName}"
"${BuildArtifactFileBaseName}.bin"
"${CG_TOOL_ROOT}/bin/armofd"
"${CG_TOOL_ROOT}/bin/armhex"
"${CCS_INSTALL_ROOT}/utils/tiobj2bin/mkhex4bin"
```

4. Generate a Tiva DFU-firmware download image file.

   In order to use the Tiva DFU upgrade utility program (dfuprog), a Tiva prefix and a DFU suffix must be attached to the binary image file. Add another post build command to accomplish this. Be sure to add this command after the previous command.

```
Project Explorer > usbserial-01 > Select
Project Properties
CCS Build > Steps (tab)
Post-build steps
```

   Add the following command on its own line.

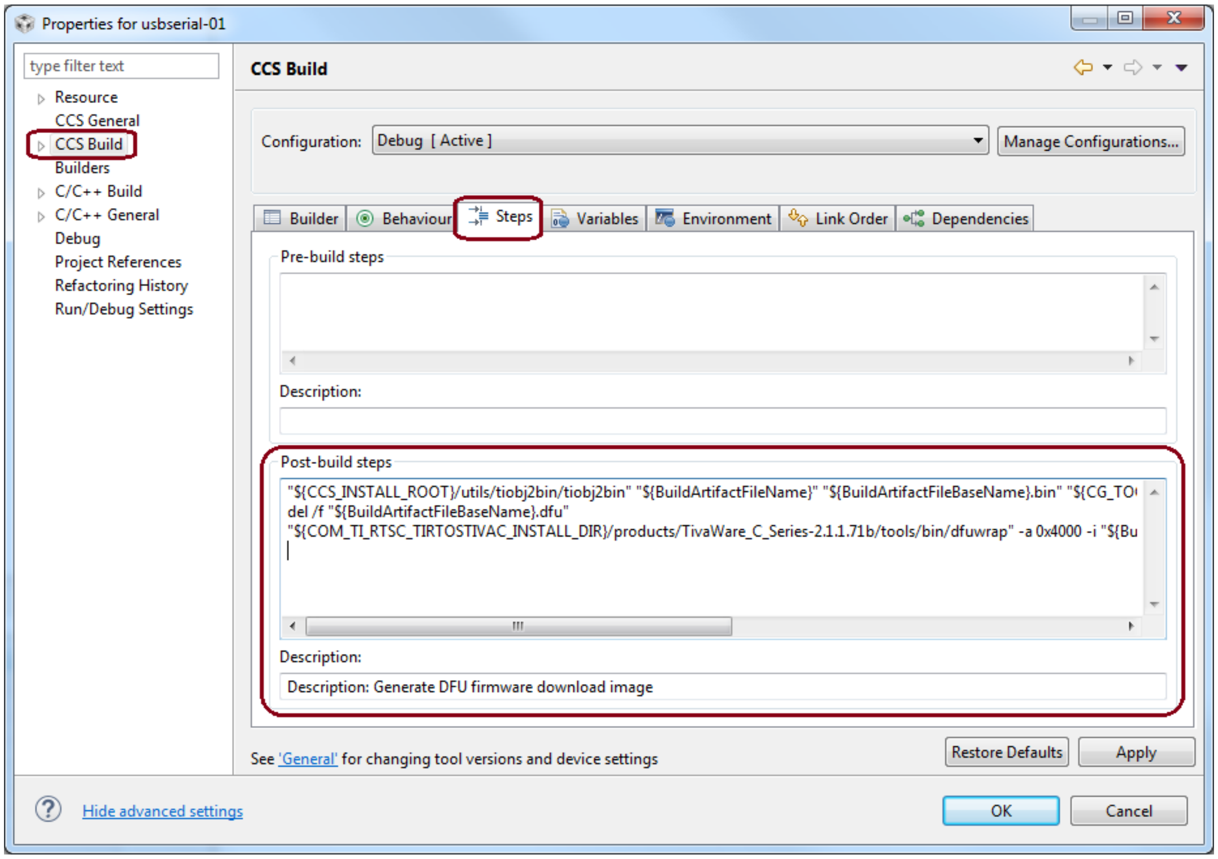
```
del /f "${BuildArtifactFileBaseName}.dfu"
```

   Add the following command, all on one line with a space separating each token.

```
"${COM_TI_RTSC_TIRTOSTIVAC_INSTALL_DIR}/products/TivaWare_C_Series-2.1.1.71b/tools/bin/dfuwrap"
-a 0x4000
-i "${BuildArtifactFileBaseName}.bin"
-o "${BuildArtifactFileBaseName}.dfu"
```

   Add the following description. You will see this description in the build console when the post build commands are executing.

```
Description: Generate DFU firmware download image
```



Configuration: Debug [Active]

Pre-build steps

Description:

Post-build steps

```
"${CCS_INSTALL_ROOT}/utils/tiobj2bin/tiobj2bin" "${BuildArtifactFileName}" "${BuildArtifactFileBaseName}.bin" "${CG_TO
del /f "${BuildArtifactFileBaseName}.dfu"
"${COM_TI_RTSC_TIRTOSTIVAC_INSTALL_DIR}/products/TivaWare_C_Series-2.1.1.71b/tools/bin/dfuwrap" -a 0x4000 -i "${Bu
```

Description: Description: Generate DFU firmware download image

See 'General' for changing tool versions and device settings

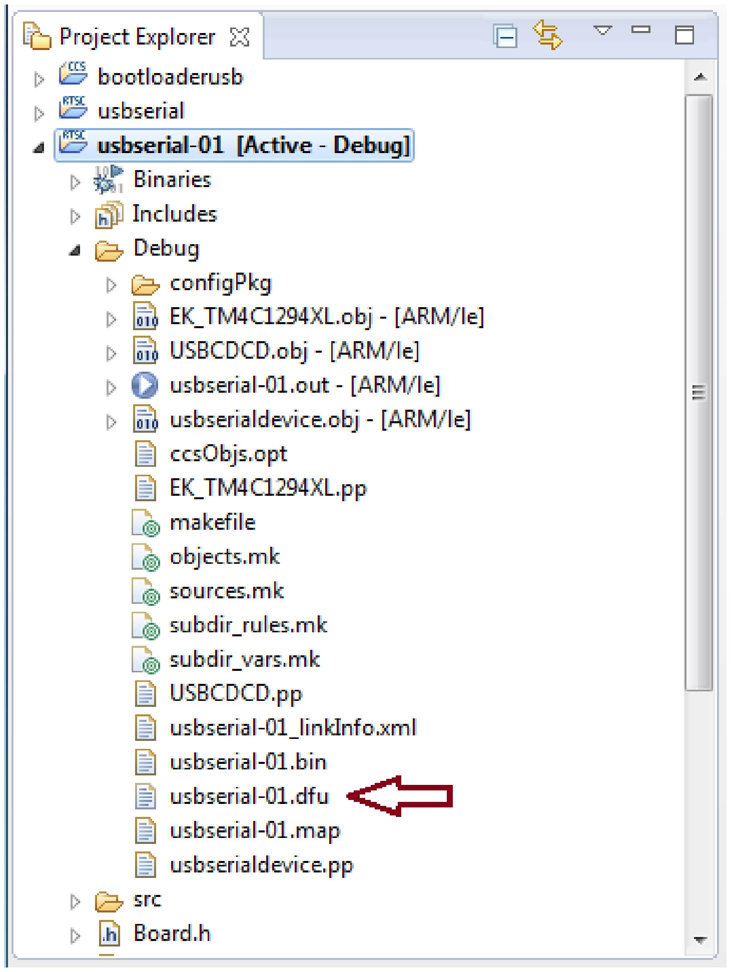
5. Build the project.

Clean and rebuild the project in order to trigger the post build steps for the first time.

```
Project Explorer > usbserial-01 > RMB > Clean Project
Project > Build Project
```

Look in the Debug folder for the Tiva DFU firmware download image file.

```
Project Explorer > usbserial-01 > Debug
usbserial.dfu
```



You have built a new firmware image which can be used for an in-field upgrade of your device. You will do this in the next part of the example.

Part 5: Firmware upgrade

You are now ready to perform an in-field firmware upgrade. You will restart your device in firmware upgrade mode, download the new firmware image using the Tiva DFU Upgrade utility, and then restart your device in runtime mode.

Make sure your evaluation kit is receiving power from the application USB receptacle (JP1 = OTG). Disconnect the USB cable from the USB debug port.

1. Restart your device in firmware upgrade mode.

   The resident boot loader on your device will sample the User 1 button at startup. If this button is pressed, the boot loader will enter USB DFU mode.

   ```
   Disconnect the cable from the application USB receptacle
   Press and hold the User 1 button while reconnecting the USB cable
   Release the button
   ```

   Your device should now be in USB DFU mode. Use the Windows device manager to locate your device. It will be listed as a Stellaris firmware upgrade device.

   ```
   +-- Stellaris Device Firmware Upgrade
        |
        +-- TivaWare Device Firmware Upgrade
   ```

   Note: You can also enter USB DFU mode by pressing the reset button.

   ```
   Press and hold the Reset button
   Press and hold the User 1 button
   Release the Reset button (keep the User 1 button pressed momentarily)
   Release the User 1 button
   ```

2. Identify your device while in upgrade mode.

   Open a Windows Command prompt. Use the TivaWare USB DFU utility program (`dfuprog`) to list all USB devices which have enumerated with the USB DFU interface.

   Note that `dfuprog` can be found in the install folder of the TivaWare C Series Windows Examples package. The default install folder is located at:

   ```
   C:/Program Files (x86)/Texas Instruments/Tiva/usb_examples
   ```

   Use the following command to list all of the USB DFU interfaces.

   ```
   dfuprog -e
   ```

   You will see output similar to the following.

```
USB Device Firmware Upgrade Example
Copyright (c) 2008-2015 Texas Instruments Incorporated.

Scanning USB buses for supported DFU devices...


<<<< Device 0 >>>>

VID: 0x1cbe    PID: 0x00ff
Device Name:   Device Firmware Upgrade
Manufacturer:  Texas Instruments
DFU Interface: <<Unknown>>
Serial Num:    0.1
Max Transfer:  1024 bytes
Mode:          DFU
TI Extensions: Supported
Target:         revision A2
Attributes:
Will Detach:       No
Manifest Tolerant: Yes
Upload Capable:    Yes
Download Capable:  Yes

Found 1 device.
```

Note: If you also have a USB cable connected to the debug port of the evaluation kit, the emulator will also be listed as a DFU device. Be careful not to upgrade this device by mistake.

Take note of the device index listed in the output (`'<<<< Device 0 >>>>'` above). You will use this index value when issuing the firmware upgrade command.

3. Upgrade the firmware on your device.

   Use the following command to upgrade the firmware on your device. This command will use the USB DFU protocol to communicate with the boot loader. Upon receiving the upgrade command, the boot loader will first erase the application portion of flash memory. It will then write the new firmware image into flash. Finally, the device will reset, which will restart it in device mode using the new firmware.

   ```
   dfuprog -i 0 -r -f usbserial-01.dfu
   ```

   After the command completes, you should again see the USB serial port device listed in the Windows Device Manager.

4. Test out the new device firmware.

   Once the device attaches to the USB bus, use the CCS Terminal view to observe the tick count you added to the transmit message. You should see output similar to the following:

   ```
   [12] TI-RTOS controls USB.
   [13] TI-RTOS controls USB.
   [14] TI-RTOS controls USB.
   ```

Congratulations. You have successfully performed an in-field firmware upgrade of your USB device. The techniques detailed in this example can be used to add firmware upgrade capabilities to most any USB device.

Part 6: USB Composite Device

A USB Composite device is a single physical device with multiple, independent interfaces. In our example, the Virtual COM Port will be one interface and the other will be the Device Firmware Upgrade (DFU) interface. This adds some complexity to the device firmware because it needs to respond to USB events on both interfaces. In addition, when switching from the runtime mode to the upgrade mode, the firmware application must be capable of restarting itself. However, compared to the previous example, this approach allows you to initiate the upgrade phase without physically accessing the device.

There are three sections below. In the first section, you will import the project and study the changes needed to construct a USB Composite device. The second section discusses the supporting INF files and host drivers. Finally, we review the runtime procedure for performing an in-field firmware upgrade.

A. Import the project and study it.

The completed CCS project is provided with the example; look in the ccs folder. Import the project **usbserialdfu** into your workspace.

```
File > Import
Code Composer Studio > CCS Projects
Next

Select search-directory: ek_tm4c1294xl_usb_serial_dfu/ccs
usbserialdfu > Select
Copy projects into workspace > Select
Finish
```

The project usbserialdfu is based on the project usbserial which you created in the preceding parts. If you compare the two projects, you will see the changes required to make a USB Composite device. These changes are explained below, file by file.

USBCDCD.c

Two new modules will be used by this file: usbddfu-rt (USB Device DFU runtime), and usbdcomp (USB Device Composite). Their respective header files are included.

```
#include <usblib/device/usbddfu-rt.h>
#include <usblib/device/usbdcomp.h>
```

USB events sent by the host are received by the firmware application in an interrupt handler. When the DFU Detach event is received, a boolean flag is used to inform the application tasks that this event has been received.

```
/* DFU upgrade request flag, set by DFU isr */
volatile bool dfuUpdateRequest = false;
```

The new DFU interface requires a callback and a runtime interface.

```
/* DFU interface event handler */
uint32_t dfuDetachCallback(void *pvCBData, uint32_t ui32Event,
        uint32_t ui32MsgData, void *pvMsgData);

/* DFU runtime interface definition */
tUSBDDFUDevice dfuDevice = {
    dfuDetachCallback,
    (void *)&dfuDevice
};
```

When creating a USB Composite device, each interface must be defined independently, and then the composite device is defined as an additional device. The composite device requires its own string descriptors.

```
/* composite interface description: "Serial with DFU" */
const unsigned char compInterfaceString[] = { ... };

/* composite configuration description: "Programmatic DFU Control" */
const unsigned char compConfigurationString[] = { ... };

/* composite descriptor string table */
const unsigned char * const compStrDesc[] = { ... };
```

The composite device requires its own event handler and device definition. Note that it also requires a new product ID. You should not reuse an existing product ID from one of its interfaces. The composite device definition also requires an array for all the device instances.

```
/* composite interface event handler */
uint32_t compHandler(void *pvCBData, uint32_t ui32Event,
        uint32_t ui32MsgData, void *pvMsgData);

/* array of devices supported by this composite device */
tCompositeEntry compDeviceAry[NUM_DEVICES];

/* top level composite device class definition */
tUSBDCompositeDevice compDevice = { ... };
```

Finally, you must allocate space for the composite device combined configuration descriptor.

```
/*  composite device combined configuration descriptor buffer */
#define DESC_BUF_SIZE  (COMPOSITE_DDFU_SIZE + COMPOSITE_DCDC_SIZE)
uint8_t compDescBuffer[DESC_BUF_SIZE];
```

When using the CDC serial device class as a composite, you must replace the initialization method with the composite version.

```
/* initialize each device instances in our composite device */
USBDCDCCompositeInit(0, &serialDevice, &(compDeviceAry[0]));
USBDDFUCompositeInit(0, &dfuDevice, &(compDeviceAry[1]));
```

Finally, you must initialize the composite device, the USB controller, and connect the device to the bus.

```
/*  Pass the USB library our device information, initialize the USB
 *  controller and connect the device to the bus.
 */
compHndl = USBDCompositeInit(0, &compDevice, DESC_BUF_SIZE, compDescBuffer);
```

<u>DFU.h</u>

This new module encapsulates the work needed to restart the device in upgrade mode. This module uses the TivaWare USB Library exclusively because the ROM code assumes complete control of the device. There are no calls to TI-RTOS. The DFU module has one public method.

```
/*
 *  ======== DFU_renumerate ========
 *  Re-enumerate the USB device in upgrade mode
 */
extern void DFU_renumerate(void);
```

usbserialdevice.c

This is the main application file. Originally, there were two tasks: one for transmitting data and one for receiving data. In this example, a third task is added (`dfuFxn`) which will initiate the switch to device firmware upgrade mode. The DFU task will wait until the other two tasks have completed. An event object is used for synchronizing the tasks. The original tasks are modified to check the DFU request flag, and to terminate when this flag is set.

The transmit task is modified to check the DFU update flag. When the flag is set, the loop is terminated and the task posts the event object.

```
Void transmitFxn(UArg arg0, UArg arg1)
{

    while (!USBCDCD_dfuUpdateRequest) {
        .../
    }

    Event_post(dfuEvent, Event_Id_00);
}
```

The receive task has the same modifications. Note that each task will post a different bit of the event object mask (Event_Id_00 and Event_Id_01).

```
Void receiveFxn(UArg arg0, UArg arg1)
{

    while (!USBCDCD_dfuUpdateRequest) {
        ...
    }

    Event_post(dfuEvent, Event_Id_01);
}
```

The DFU task simply waits until the other two tasks have posted the event object. Then it invokes the switch to upgrade mode. Note that the ROM code will use the stack provided by the DFU task.

```
void dfuFxn(UArg arg0, UArg arg1)
{
    UInt mask;

    /* wait for other tasks to finish */
    mask = Event_Id_00 | Event_Id_01;
    Event_pend(dfuEvent, mask, Event_Id_NONE, BIOS_WAIT_FOREVER);

    /* re-enumerate in dfu mode */
    DFU_renumerate();
}
```

The event object is created in the function main.

```
int main(void)
{
    ...

    /* event object used to signal dfu mode */
    dfuEvent = Event_create(NULL, NULL);

    ...
}
```

usbserialdevice.cfg

The configuration file declares all modules used by the application. This information is used when creating the list of libraries for the linker. The application added the Event module, so the configuration file must declare the use of this module.

```
var Event = xdc.useModule('ti.sysbios.knl.Event');
```

In this example, the tasks are created statically. The DFU task is also created statically. Note that the DFU task priority is lower than the other tasks. This ensures that the DFU task does not run until the other tasks have completed their termination sequence (which happens after they post the event object).

```
/* dfu task */
var taskParams = new Task.Params();
taskParams.instance.name = "dfu";
taskParams.stackSize = 512;
taskParams.priority = 1;
Program.global.receive = Task.create("&dfuFxn", taskParams);
```

B. Host support files

The first time you connect your evaluation kit to the USB bus using this project as the firmware, it will enumerate with a Vendor ID of 1CBE and a Product ID of 0102. The Windows Device Manager will prompt you for the driver files. Use the Windows folder contained within this CCS project.

```
usbserialdfu/Windows
```

The firmware will enumerate as a composite device, so you will be prompted two times to install drivers, once for each device. Use the same folder above both times.

Each device of your composite enumerates with the same Vendor ID and Product ID but with different Interface IDs. The Windows folder contains two INF files, one for each interface.

```
usb_serial_port.inf    USB\VID_1CBE&PID_0102&MI_00
usb_serial_dfu.inf     USB\VID_1CBE&PID_0102&MI_02
```

This example uses the USB Communication Device Class for the serial port. Microsoft Windows provides a host driver for this class. The `usb_serial_port.inf` file specifies to use this driver.

This example also uses the USB Device Firmware Upgrade Class but Microsoft Windows does not provide a driver for this. However, the TivaWare C Series product provides a Windows host driver. These are included in the amd64 and i386 folders (they can also be found in the TI-RTOS installation). The `usb_serial_dfu.inf` file instructs Microsoft Windows to install the TivaWare C Series host drivers.

When developing your own USB devices, you will need to modify these INF files to specify your own Product ID, Vendor ID, and Interface IDs.

C. Firmware upgrade procedure

As a prerequisite, you must build this project and download it to your device using CCS. Refer back to Part 3 above for instructions.

To complete the steps below, you will need a firmware image file. You can use the DFU file generated when building this project. Keep in mind that when using this image file, it will be identical to the firmware already on the target. After you perform the upgrade, you will not see any runtime differences.

We assume that your device is running in USB runtime mode.

Note: If you have the emulator connected to your host, unplug it now.

You will use the `dfuprog` program to switch your device to upgrade mode. Once your device has re-enumerated in DFU mode, you will use the `dfuprog` program to download the new firmware image and restart your device.

1. Identify your device.

   Open a Windows Command prompt. Use the TivaWare USB DFU utility program (`dfuprog`) to list all USB devices which have enumerated with the USB DFU interface.

   Use the following command to list all of the USB DFU interfaces.

```
dfuprog -e
```

   You will see output similar to the following.

```
USB Device Firmware Upgrade Example
Copyright (c) 2008-2015 Texas Instruments Incorporated.  All rights reserved.

Scanning USB buses for supported DFU devices...


<<<< Device 0 >>>>

VID: 0x1cbe    PID: 0x0102
Device Name:   Virtual COM Port
Manufacturer:  Texas Instruments
DFU Interface: <<Unknown>>
Serial Num:    12345678
Max Transfer:  1024 bytes
Mode:          Runtime
Attributes:
   Will Detach:       Yes
   Manifest Tolerant: Yes
   Upload Capable:    Yes
   Download Capable:  Yes

Found 1 device.
```

   Note: If you also have a USB cable connected to the debug port of the evaluation kit, the emulator will also be listed as a DFU device. Be careful not to upgrade the emulator device by mistake.

   Take note of the device index listed in the output (`'<<<< Device 0 >>>>'` above). You will use this index value when issuing the firmware upgrade command.

2. Switch your device to DFU mode.

   Use the following command to re-enumerate your device in DFU mode. This will temporarily remove your device from the USB bus and then re-attach as a new device.

```
dfuprog –i 0 –m
```

List the USB devices again.

```
dfuprog -e
```

This time, you will see the following output. Note the new product ID and device name indicating your device has re-enumerated in DFU mode.

```
USB Device Firmware Upgrade Example
Copyright (c) 2008-2015 Texas Instruments Incorporated.  All rights reserved.

Scanning USB buses for supported DFU devices...


<<<< Device 0 >>>>

VID: 0x1cbe     PID: 0x00ff
Device Name:    Tiva Device Firmware Update
Manufacturer:   Texas Instruments Incorporated
DFU Interface: <<Unknown>>
Serial Num:     00000000
Max Transfer:   1024 bytes
Mode:           DFU
TI Extensions: Supported
Target:          revision A2
Attributes:
   Will Detach:       No
   Manifest Tolerant: Yes
   Upload Capable:    Yes
   Download Capable:  Yes

Found 1 device.
```

Be mindful of the device index. Depending on how many DFU capable devices are connected, the device index of you device might change when switching to DFU mode. Be sure to use the correct device index in the next step.

3. Upgrade the firmware on your device.

   Use the following command to upgrade the firmware on your device. The boot loader will erase the application portion of flash memory, and then write the new firmware image into flash. It will then restart the device in runtime mode using the new firmware.

```
dfuprog –i 0 –r –f usbserialdfu.dfu
```

Congratulations. You have completed the USB Device Firmware Upgrade example. Thank you for supporting Texas Instruments Incorporated.