Freescale USB Device Stack
Users Guide

Document Number:USBUG
Rev. 12
05/2012

freescale™
semiconductor

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: USBUG
Rev. 12
05/2012

Revision history

To provide the most up-to-date information, the revision of our documents on the World Wide Web will be the most current. Your printed copy may be an earlier revision. To verify you have the latest information available, refer to:

http://www.freescale.com

The following revision history table summarizes changes contained in this document.

Revision Number	Revision Date	Description of Changes
Rev. 1	05/2009	Alpha Customer Release.
Rev. 2	05/2009	Added CDC feature description.
Rev. 3	06/2009	• Added ColdFire V1 support and PHDC Multi-Specialization Device Demo application. • Changed USB to Serial Demo name to Virtual Communication (COM) Demo.
Rev. 4	09/2009	Launch release. Customized for Medical Applications.
Rev. 5	10/2009	Added SD Card demo application.
Rev. 6	04/2010	Updated USB Stack installation and uninstallation and Medical Applications USB Stack directory structure diagram to add support for S08MM128, S08JE128, MCF51MM256, and MCF51JE256 devices.
Rev. 7	06/2010	• Added support for CFV2 devices. • Rebranded Medical Applications USB Stack to Freescale USB Stack with PHDC.
Rev. 8	09/2010	• Added support for CodeWarrior 10 • Added USB audio demo application • Fig 2-1:Freescale USB stack with PHDC Directory Structure updated
Rev. 9	01/2011	• Added USB DFU demo application • Update images in various demo application • Minor editorial changes
Rev. 10	07/2011	• Added battery charging demo application • USB FATFS User Guide incorporated in the USB User Guide
Rev. 11	03/2012	• Deleted chapters “FAT File System” and “AppendixJ_FATFS_Demo_Test” • Replaced the term "Freescale USB Stack with PHDC" with "Freescale USB Stack" • Updated Installer screenshots • Editorial Changes

Revision Number	Revision Date	Description of Changes
Rev. 12	05/2012	Added • Appendix J: Video Device Class Demo Applications • Appendix K: MSD and CDC Composite Demo • Appendix L: HID Audio Video Composite Demo

Chapter 1
Before You Begin

Chapter 2
Getting Familiar

Chapter 3
Freescale USB Stack Architecture

Chapter 4
Developing New Class Drivers

Chapter 5
Developing Applications

Appendix A
Working with the Software

Appendix B
Human Interface Device (HID) Demo

Appendix C
Personal Healthcare — Multi-Specialization Device Demo

Appendix D
Human Interface Device (HID) Demo

Appendix E
Personal Healthcare – Weigh Scale Device Demo

Appendix F
SD Card Demo

Appendix G
USB Audio Demo

Appendix H
DFU Class Demo

Appendix I
Battery Charging Device Demo Application

Appendix J
Video Device Class Demo Applications

Appendix K
MSD and CDC Composite Demo

Appendix L
HID Audio Video Composite Demo

Chapter 1 Before You Begin

1.1 About Freescale USB Stack

Universal Serial Bus commonly known as USB is a serial bus protocol that can be used to connect external devices to the host computer. In today's world, it is one of the most popular interfaces connecting devices such as microphone, keyboards, storage devices, cameras, printers, and many more. USB interconnects are also getting more and more popular in the medical segments. The Freescale USB Stack enables you to use Freescale 8-bit, 16-bit, and 32-bit MCUs (for example: Kinetis k40, S08, CFV1, and so on) silicon to make the devices listed above.

It abstracts the details of Kinetis k40, S08, CFV1, and CFV2 devices, and the USB IP used. It provides a higher level interface to the application. The application developers only need to concentrate on the application in hand without worrying about the USB details.

1.2 About this book

This book describes the Freescale USB Stack architecture. Table 1-1 shows the summary of chapters included in this book.

Table 1-1. USBUG summary

Chapter Title	Description
Before you begin	This chapter provides the prerequisites of reading this book.
Getting Familiar	This chapter provides the information about the Freescale USB Stack software suite.
USB Stack Architecture	This chapter discusses the architecture design of the Freescale USB suite.
Working with the Software	This chapter provides information on how to build, run, and debug drivers and applications.
Developing Class Drivers	This chapter discusses the steps a developer must take to develop applications on top of the pre developed classes.
Human Interface Device (HID) Demo	This chapter provides the setup and running HID demo using USB stack – Kinetis k40, S08, CFV1, and CFV2 devices are used as examples.
Personal Healthcare – Weigh Scale Device Demo	This chapter provides the setup and running Personal Healthcare – Weigh Scale device demo using USB stack – Kinetis k40, S08, CFV1, and CFV2 devices are used as examples.
Virtual Communication (COM) Demo	This chapter provides the setup and running Communication Device Class (CDC) demo using USB stack – Kinetis k40, S08, CFV1, and CFV2 devices are used as examples.
Personal Healthcare – Multi-Specialization Device Demo	This chapter provides the setup and running Personal Healthcare – Multi-Specialization device demo using USB stack – Kinetis k40, S08, CFV1, and CFV2 devices are used as examples.

SD Card Demo	This chapter provides the setup and running Mass Storage Class (MSC) demo using USB stack – Kinetis k40, S08, CFV1, and CFV2 devices are used as examples.
USB Audio Demo	This chapter provides information about Audio Demos – how to run Audio Speaker Demo and Audio Generator Demo.
DFU Class Demo	This chapter provides information about DFU Class Demo – how to setup the DFU Class Demo and how to run the DFU Class Demo.
Battery Charging Demo Application	This chapter provides the setup and running the Battery Charging Demo Application using USB stack – Kinetis K40, K53 and K60, CF+
FATFS Demo and Test application	This chapter provides the setup and running USB FATFS demo example and USB FATFS test example for CFV1 processors.

1.3 Reference material

Use this book in conjunction with:

- *Freescale USB Stack Device API Reference Manual* (document USBAPIRM)
- *Freescale USB Stack Host Users Guide* (document USBHOSTUG)
- S08 USB Device Source Code
- ColdFire V1 USB Device Source Code
- ColdFire V2 USB Device Source Code
- USB Audio Class API Reference Manual
- USB Video Class API Reference Manual
- USB MCD Class API Reference Manual
- USB CDC Class API Reference Manual
- USB BM device stack source
- Application Note: Audio Reproduction on HCS12 Microcontrollers (AN2250), Rev. 0, 2002
- File Allocation Table information at ***http://en.wikipedia.org/wiki/File_Allocation_Table***
- FATFS Module Application Note at ***http://elm-chan.org/fsw/ff/en/appnote.html***
- USB Host source code.
- USB FATFS source code
- USB HID Class API Reference Manual
- USB BM device stack source code

For better understanding, refer to the following documents:

- USB Specification Revision 1.1
- USB Specification Revision 2.0
- USB Device Class Definition for Audio Devices Revision 1.1
- USB Device Class Definition for Video Devices Revision 1.0a
- S08 Core Reference
- ColdFire V2 Core Reference
- ColdFire V1 Core Reference
- CodeWarrior Help
- MCF51JM128 Reference Manual
- USB Device Class Definition for DFU Devices Revision 1.1 (also called DFU class specificatio).
- MCF52259 Reference Manual
- Battery Charging Specification Rev 1.1
- K60 Sub-Family Reference Manual

1.4 Acronyms and abbreviations

API	Application programming Interface
CDC	Communication Device Class
CDP	Charging Downstream Port
CFV1	ColdFire V1 (MCF51JM128 CFV1 device is used in this document)
CFV2	ColdFire V2 (MCF52221, MCF52259, and MCF52277 CFV2 device is used in this document)
COM	Communication
DBCS	Double-Byte Character Set
DCP	Dedicated Charging Port
EVB	Evaluation
DFU	Device Firmware Upgrade
FAT	File Allocation Table
FATFS	File Allocation Table file system
HCI	Host Controller Interface
HID	Human Interface Device
IDE	Integrated Development Environment
JM60	MC9S08JM60 Device
JM16	MC9S08JM16 Device
JM128	MCFJM128 Device
JS16	MC9S08JS16 Device
K60	MK60N512VMD Device
MBR	Master Boot Record
MSD	Mass Storage Device
MSC	Mass Storage Class
M52259	MCF52259 Device
OEM	Original Equipment Manufacturer
PC	Personal Computer
PD	Portable Device
PHD	Personal Healthcare Device
PHDC	Personal Healthcare Device Class
SCSI	Small Computer System Interface
SDP	Standard Downstream Port
USB	Universal Serial Bus

1.5 Important terms

Table 1-2 shows the terms used throughout the book.

Table 1-2. Important terms

Term	Description
Class Driver	These are the high level function specific drivers that can control large number of different devices of a similar type.
Code Page	Code page is another name for character encoding. It consists of a table of values that describes the character set for a particular language.
Cluster	To reduce the overhead of managing on-disk data structures, the file system does not allocate individual sectors, but contiguous groups of sectors, called clusters.
Continua Alliance	This is a consortium of companies to establish standards for the medical segment devices.
Expansion Card	This is the card where the silicon is embedded and can be loaded on to the hardware board.
DemoJM	This is the physical hardware where the expansion card with the silicon is mounted.
Enumeration	It is a process in the USB protocol by which the host identifies the devices connected to it.
FAT12	A type of FAT file system that uses 12 bits value to address clusters.
FAT16	A type of FAT file system that uses 16 bits value to address clusters.
FAT32	A type of FAT file system that uses 32 bits value (in which 4 bits are reserved) to address clusters.
Kinetis k40, S08, CFV1, and CFV2 Processors	These are low-end family of processors provided by the Freescale.
Long File Name	In a file system that supports long file names, a file or directory name can be as long as 255 characters including one or more dots and extensions. A complete path of the file has a maximum of 260 characters, so volumes with many levels of directories must use shorter names.
USB Low Level Drivers	USB low level drivers are the driver software layers that interface the hardware and abstracts them for the class drivers.
USB Chapter 9 Requests	These are the framework requests made by the host to the device that the device must respond to. These are defined in Chapter 9 of the USB specification document.

Table 1-2. Important terms (continued)

Term	Description
Attach versus Connect	A downstream device is considered to be attached to an upstream port when there is a physical USB cable between them. A downstream device is considered to be connected to an upstream port when there is attached to the upstream port and when the downstream device has pulled either the D+ or D- data line high through a 1.5 K resistor in oeder to operate either as a low or full speed device.
Downstream Port	A Downstream Port refers to either a Standard Downstream Port (SDP) or a Charging Downstream Port (CDP).
USB Charger	A USB Charger is a device with Dedicated Charging Port, such as a wall adapter or car power adapter.
Portable Device	A Portable Device is considered to be any USB or OTG device that is capable of operating from its own battery and it also capable of drawing current from the USB port for its purposes of operating and/or charging its battery.
Weak Battery Threshold	Minimum voltage charge level of a battery such that above this threshold the device is considered to function normally.
Dead Battery Threshold	Maximum charge level of a battery such that below this threshold the device is assumed to not been able to function anymore.
Sector	Sector is the smallest storage unit in a mass storage medium. Typically, a sector holds 512 bytes of information. However, some medium can have sector size more than 512 bytes.
Partition	A partition is a logical division on mass storage device. The term is also known as Volume or Logical Disk.

Chapter 2 Getting Familiar

2.1 Introduction

The Freescale USB Stack device contains the low level driver code, commonly used class drivers, and some basic applications. This document intends to help you develop an understanding of the stack and to assist you in developing more classes and applications. The document is targeted for firmware application developers who would like to develop the applications using USB as the transport.

2.2 Software suite

The software suite comprises of the USB low level drivers for the Kinetis k40, S08, CFV1, and CFV2 families, generic class drivers, and applications. The class drivers are programmed with generic code, so they can be used with other processors like CFV1 and CFV2 without a line of code change if the low level drivers comply with the driver interface.

2.3 Directory structure

The software suite has a standard directory structure. You can extend it easily to accommodate more applications, classes, and low level drivers for different processor families.

Figure 2-1 shows the directory structure.

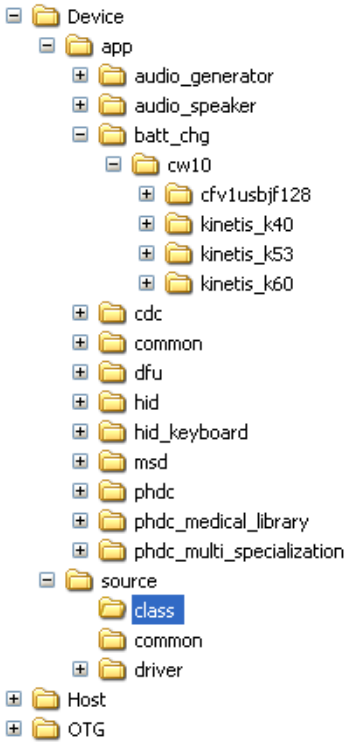


Figure 2-1. Freescale USB Stack directory structure

Chapter 3 Freescale USB Stack Architecture

3.1 Architecture overview

Figure 3-1 shows the Freescale USB Stack architecture.

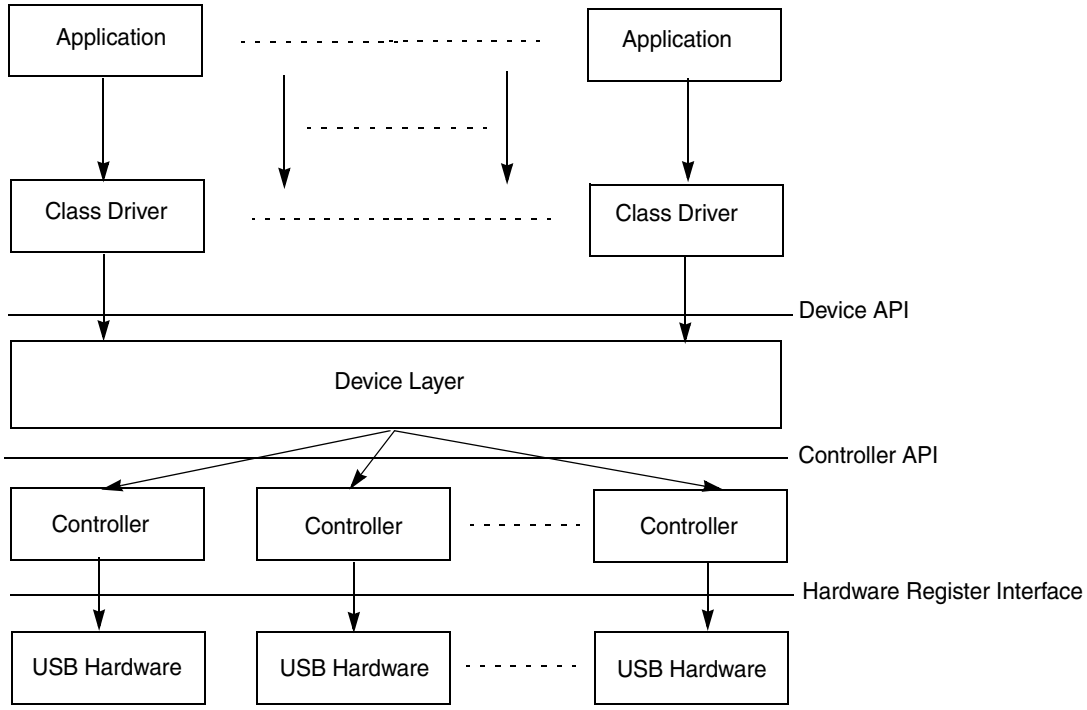


Figure 3-1. Freescale USB Stack Architecture

The USB stack is mainly divided into three layers with the applications being developed on top of them. The layered architecture helps the application developers concentrate on developing the application without being concerned about the other layers. The applications can also be seamlessly ported over to other cores after the low level driver for that core is available.

The class driver layers implement various class drivers that have different functions. The *USB chapter 9* requests are also part of the functionality of the class driver layer. These are implemented as common module and can be used as is to develop new classes. Some of the examples here are storage, human interface device, personal healthcare device, and so on.

This driver interfaces with the device layer for its lower layer functions. For some functions, the device layers do not provide additional functionality and call the lower layer functions. Most of the validation for parameters is done in this layer. This layer must be independent of any underlying hardware and therefore can be ported on the different hardware platform with minimal changes.

The device layer can sit on top of the controller layer that is the hardware dependent layer and interfaces the hardware registers.

As stated earlier, the layered architecture helps the application developers to develop applications. However, it does not limit the developer to interface lower layer APIs if they prefer to.

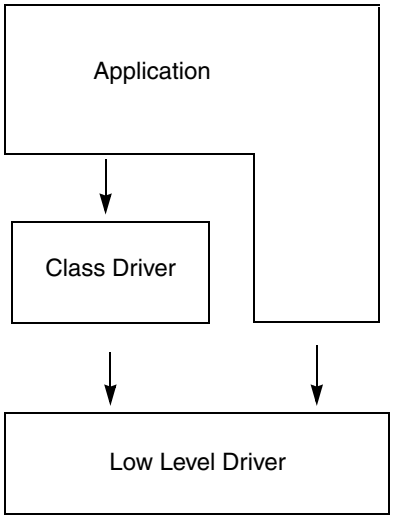


Figure 3-2. Kinetis k40, S08, CFV1, and CFV2 USB Stack Architecture Layers

CAUTION

Simultaneous use of driver APIs and class APIs may have undefined behavior. In this case, the driver functionality will not work as defined in this document.

3.2 Software flows

This section describes the execution flow of the stack across various layers and modules.

3.2.1 Initialization flow

Figure 3-3 describes stack initialization flow.

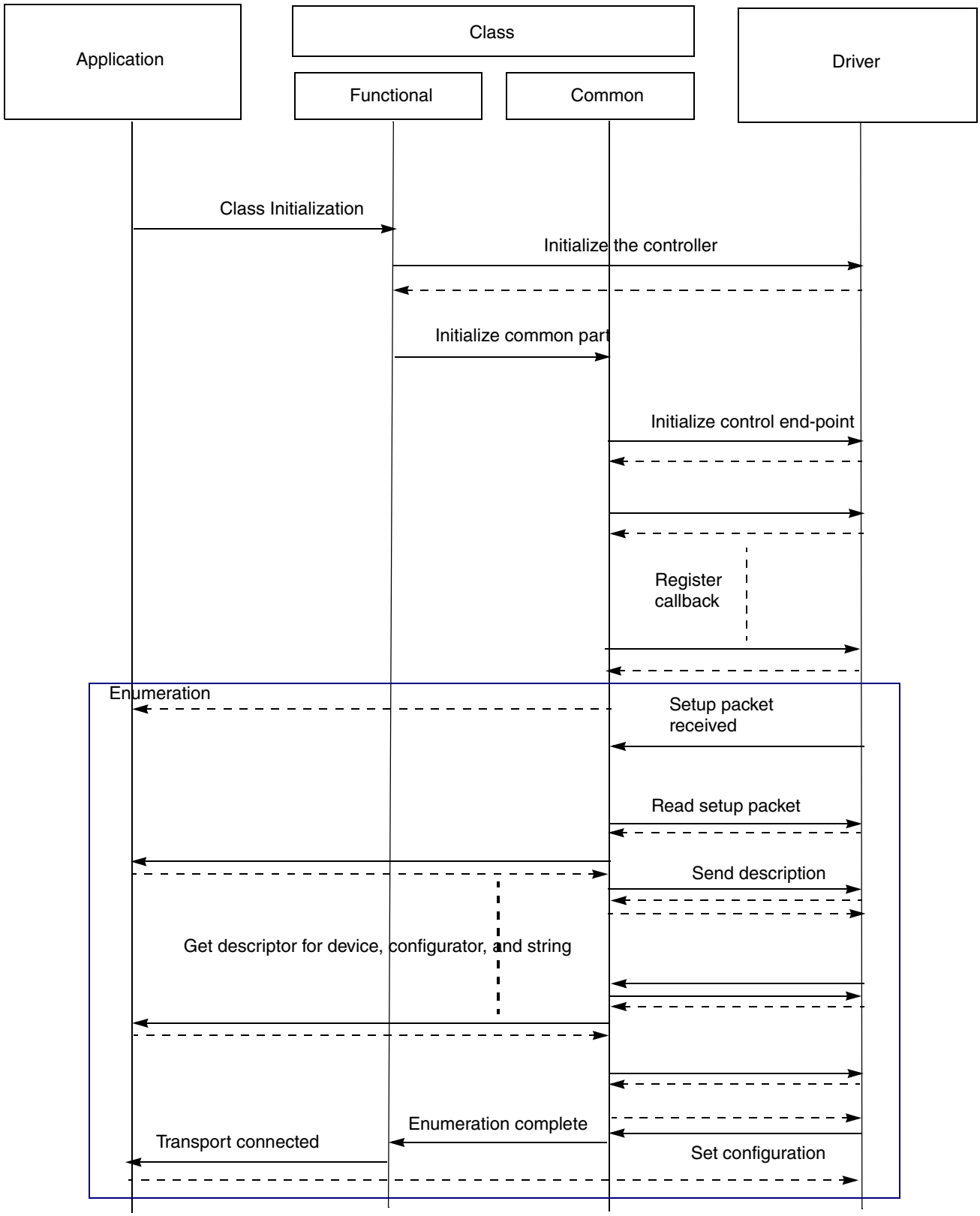


Figure 3-3. Sequence diagram for stack initiation

The initialization flow starts when the application initializes the class driver that in turn initializes the low level driver and the controller. The class driver also registers the callbacks it requires for events occurring in the USB bus. Sometime after this, the host starts the enumeration process by sending the setup packet to get descriptors for device, configuration, and string. These requests are handled by the class driver that uses the descriptors defined by the application. The enumeration finally ends when the host sets the device configuration. At this point, the class driver notifies the application that the connection has been established.

3.2.2 De-initialization flow

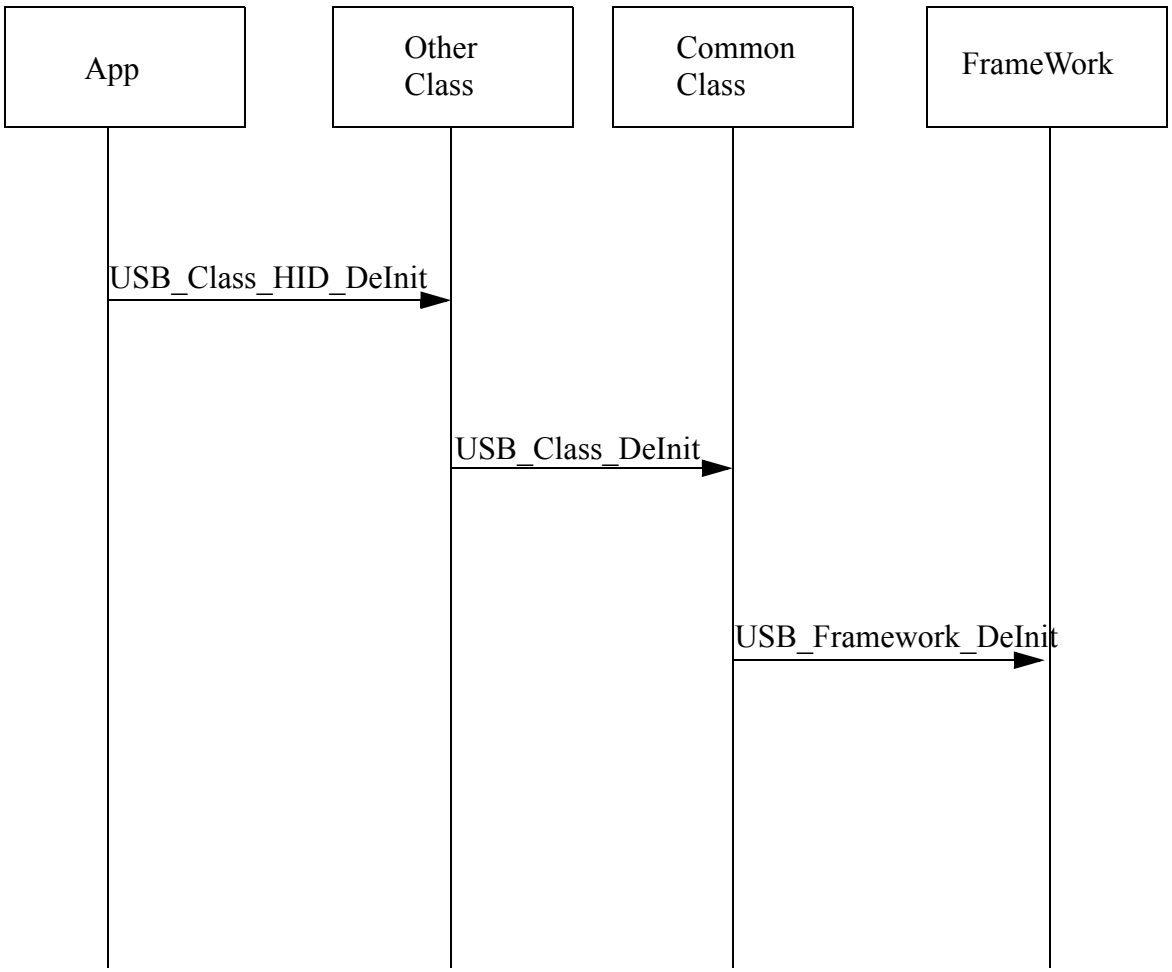


Figure 3-4. Sequence diagram for stack de-initiation

3.2.3 Transmission flow

Figure 3-5 describes stack transmission flow.

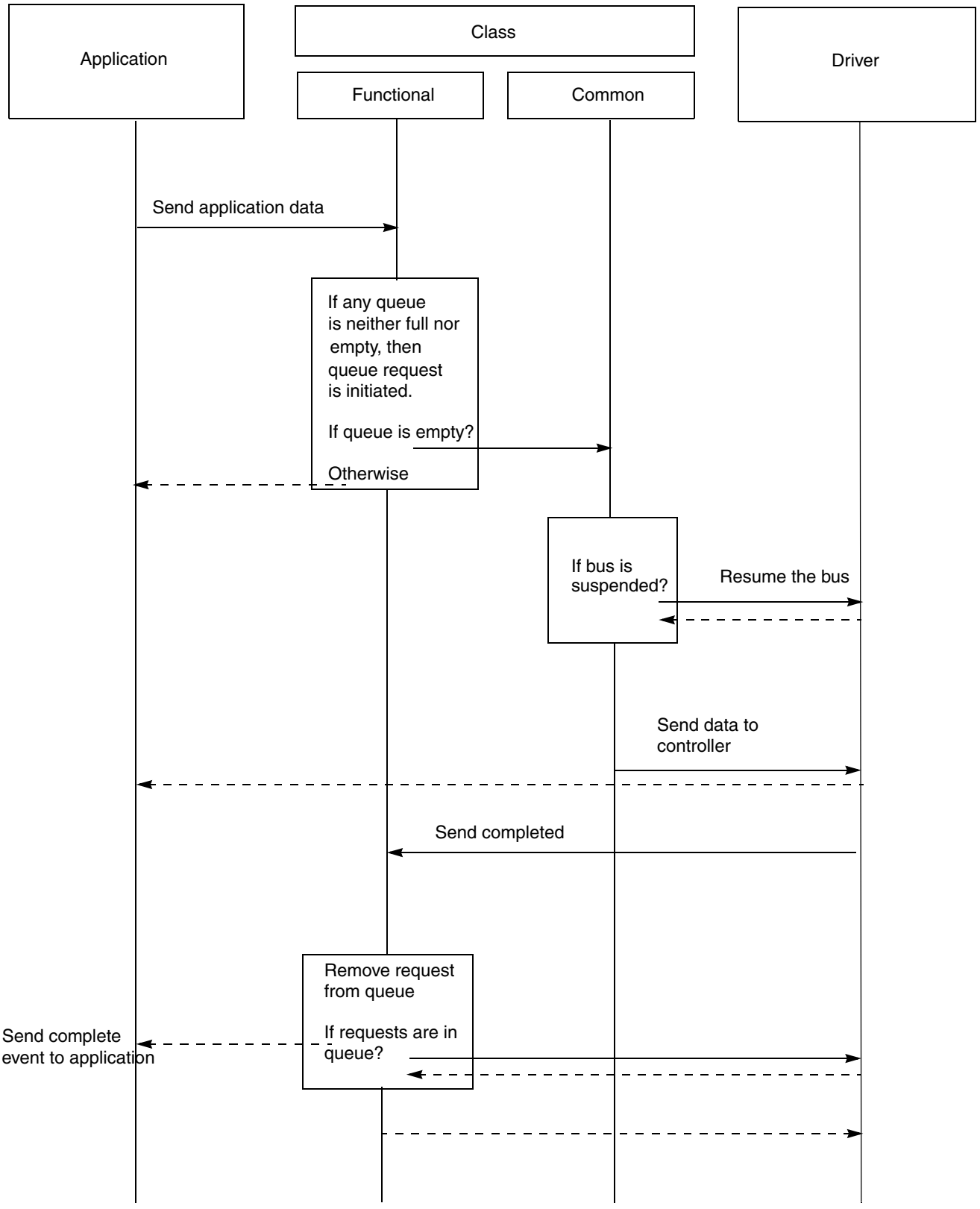


Figure 3-5. Sequence diagram for stack transmission

The application transmits data to the USB host by calling the class driver specific send API. The class driver checks for the current status of the queue. If the queue is full then the function returns with a BUSY status, if there is already an outstanding request that has not been completed yet, it queues the request unless the queue is empty, it prepares to pass the request to the low level driver. As part of the preparation, it checks whether the bus is suspended or not. Incase the bus is suspended, it wakes the bus and the bus then sends the request to the lower layer driver. When the send API operation is completed, the class driver removes the request from the queue and sends the next in queue if it exists.

3.2.4 Reception flow

Figure 3-6 below describes stack reception flow.

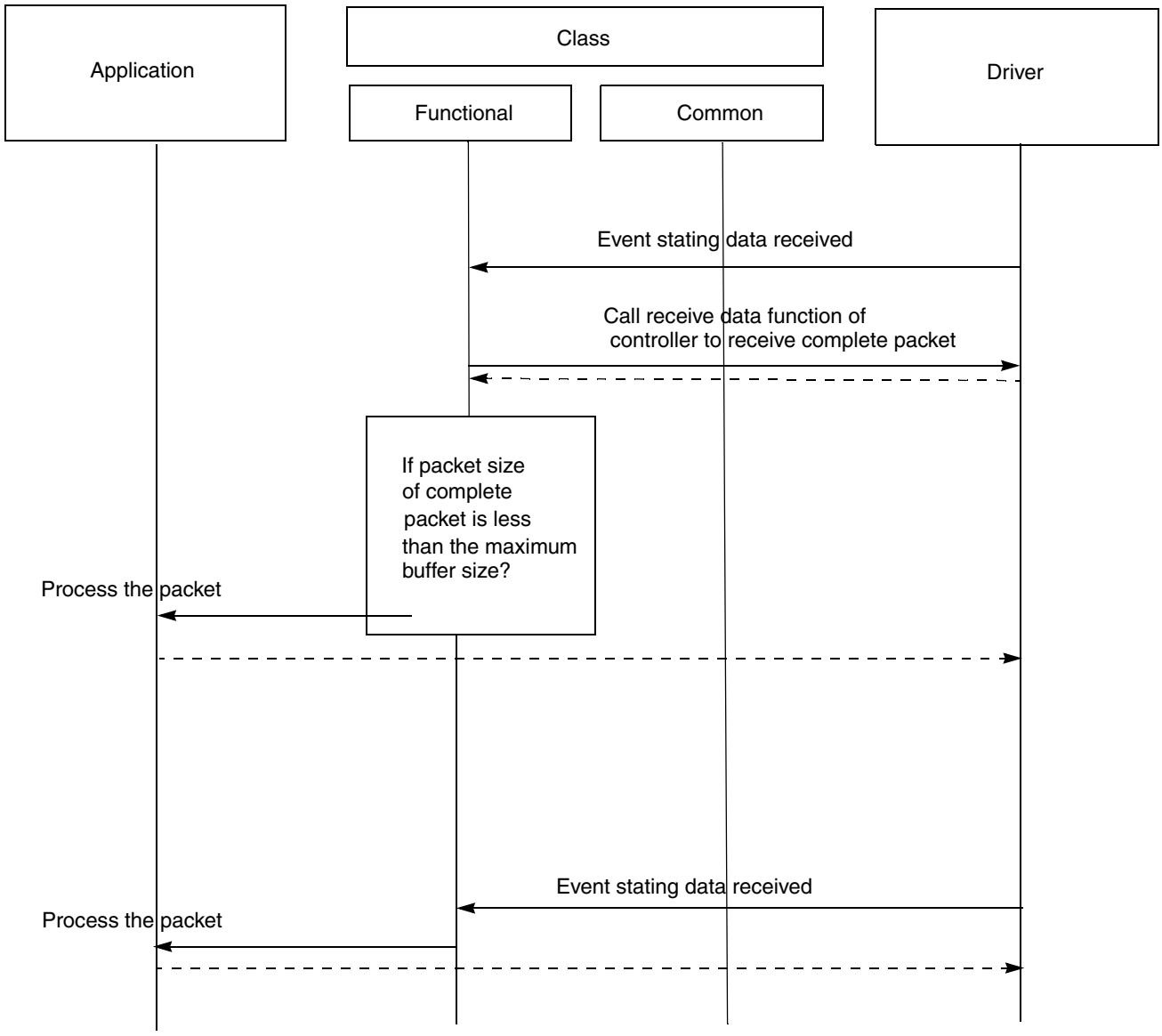


Figure 3-6. Sequence diagram for stack reception

When the controller receives the data on its receiver port, the low level driver sends an event to the class driver. The class driver calls the low level driver to receive the packet in its buffers. If the size of the packet is smaller than the size of the end-point buffer, then the class driver processes the packet immediately. If the size of the packet is greater than the endpoint buffer size, the class driver waits for an event from the low level driver with the complete packet. The class driver processes the packet after it is received.

Chapter 4 Developing New Class Drivers

4.1 Introduction

This chapter provides user methodology for developing new applications based on existing available class drivers. It also describes how a user can develop a new class driver and application using USB low level stack framework.

Support for HID, CDC, and PHDC class drivers is already implemented in the package. Refer *Freescale USB Stack Device API Reference Manual* (document USBAPIRM.pdf).

Before starting with developing the class drivers, Figure 4-1 shows the current design of the class drivers.

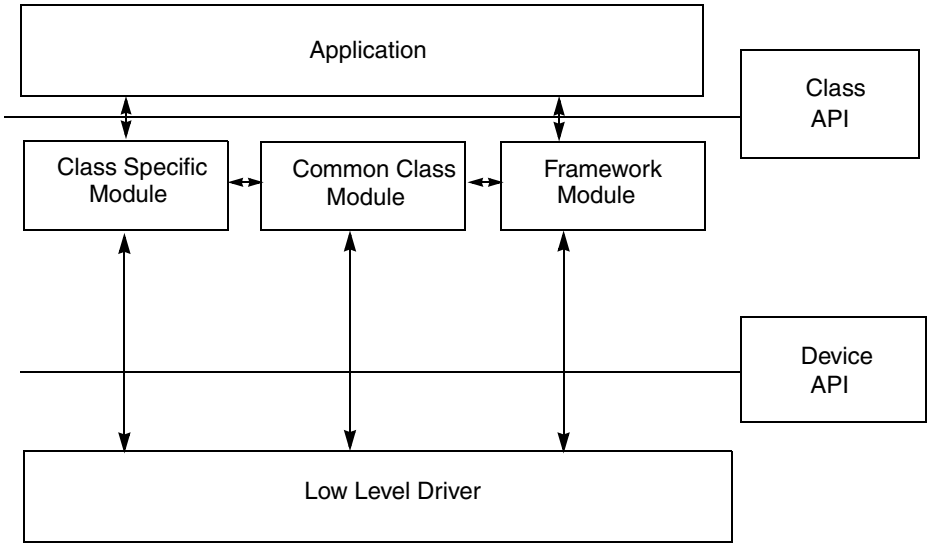


Figure 4-1. Current design of class drivers

The class drivers are divided into three modules.

- Framework Module—The framework module handles all requests to the control end point. It implements all responses to the *USB Chapter 9* requests. It interacts with the application to get the USB descriptor information.
- Common Class Module—The common class module contains implementation independent to application specific classes. It handles functions like suspend/resume, reset, stall, and SOF that needs to be present for all classes.
- Class Specific Module—This module implements class specific functionality. It implements all interactions with non control end points. The data sent and received on these end points are class specific. This module also implements the class specific requests on the control end point.

Although, a developer could start developing a new class by using the device API and implementing complete functionality. However, it is recommended to use a similar design like the existing classes and reusing some pre-existing common modules like the common class module and the framework module.

4.2 Steps for developing new class drivers

This section explains how a user can develop new class drivers based on an existing USB low level device framework and common class specific module.

4.2.1 Directory structure

Define the class API as shown in Figure 4-1. The application must use this interface to call the class driver. These can be similarly defined like the pre-existing hid and phdc classes. The interface definitions are in usb_hid.h and usb_phdc.h and so on in the /source/class directory.

Figure 4-2. Class Directory Structure

Staying in the same directory creates a class specific module as shown in Figure 4-1. This can be done by creating two files namely usb_<newclass>.c and usb_<newclass>.h. Implement the class specific code in the .c file and put the interface definition in the .h file.

4.2.2 Class initialization

Implement class initialization function. This function initializes class specific data structures. This function also initializes USB Common Class Module and USB Device Framework through USB_Class_Init() and USB_Device_Init() respectively.

Typically, three callbacks are provided by an application for interfacing with class driver.

- Class callback (to receive various USB bus events)
- To support vendor specific requests.
- To handle class specific requests (application specific like string descriptor handling etc.)

Pseudo Code:

```
uint_8 USB_Class_XYZ_Init (
      uint_8    controller_ID,           /* [IN] Controller ID */
      USB_CLASS_CALLBACK class_callback, /* [IN] Class Callback */
      USB_REQ_FUNC vendor_req_callback,  /* [IN] Vendor Request Callback */
      USB_CLASS_SPECIFIC_HANDLER_FUNC param_callback /* [IN] Class Specific requests Callback
      */
)
{
      uint_8 index;
      USB_ENDPOINTS *ep_desc_data = (USB_ENDPOINTS *)
          USB_Desc_Get_Endpoints(controller_ID);

      /* Initialize the device layer*/
      USB_Device_Init(controller_ID, ep_desc_data->count+1);
      /* Initialize the generic class functions */
      USB_Class_Init(controller_ID,
          USB_Class_XYZ_Event, USB_Class_XYZ_Requests);

      <Class Specific Initialization code goes here >

      /* save the XYZ class callback pointer */
      g_class_callback = class_callback;

      /* save the vendor request callback pointer */
      g_vendor_req_callback = vendor_req_callback;

      /* Save the callback to ask application for class specific
         params*/
      g_param_callback = param_callback;
}
```

4.2.3 Class callback routine

This routine is called by USB Common Class Module to notify class driver about various USB events. The following events are notified:

- **USB Bus Reset**

  This event is notified when USB Bus Reset is detected by Device Controller. The class driver should reset its data structure after receiving this event. Depending on the class requirement, the event can be propagated to application through a callback.
- **Enumeration Complete**

  This event is notified when USB Bus Enumeration is completed and Set Configuration call is received from USB host. Class driver should now initialize all the endpoints (USB_Device_Init_EndPoint()) other than control endpoint. It should also register callback functions (USB_Device_Register_Service()) to handle endpoint events and set Endpoint Status as “idle” (USB_Device_Set_Status()).
- **Configuration Change**

  This event is notified when SET CONFIGURATION call is received from USB host.

  Once this event is received, Enumeration Complete event is notified to the class driver.
- **Data Send Complete**

  This event is notified when data is sent through an endpoint.
- **Data Received**

  This event is notified when data is received on an endpoint.

Pseudo Code:

```
static void USB_Class_XYZ_Event (
      uint_8 controller_ID,   /* [IN] Controller ID */
      uint_8 event,           /* [IN] Event Type */
      void* val               /* [IN] Pointer to configuration Value */
)
{
      uint_8 index;

      if(event == USB_APP_ENUM_COMPLETE)
      {
          uint_8 count = 0;
          /* get the endpoints from the descriptor module */
          USB_ENDPOINTS *ep_desc_data = (USB_ENDPOINTS *)
          USB_Desc_Get_Endpoints(controller_ID);

          /* intialize all non control endpoints */
          while(count < ep_desc_data->count)
          {
              USB_EP_STRUCT_PTR ep_struct=
              (USB_EP_STRUCT_PTR)&ep_desc_data->ep[count];

              (void)USB_Device_Init_EndPoint(controller_ID,
              ep_struct, TRUE);

              /* register callback service for the endpoint */
              (void)USB_Device_Register_Service(controller_ID,
              (uint_8)(USB_SERVICE_EP0+ep_struct->ep_num),
              USB_Class_XYZ_Service_Endpoint);


              /* set the EndPoint Status as Idle in the device layer */
              (void)USB_Device_Set_Status(controller_ID,
              (uint_8)(USB_STATUS_ENDPOINT | <ENDPOINT NUMBER> |
              (ep_struct->direction << USB_COMPONENT_DIRECTION_SHIFT)),
              USB_STATUS_IDLE);

              count++;
          }
      }
      else if(event == USB_APP_BUS_RESET)
      {
          <Re-Initialize Class Specific Data Structure >
      }
      if(g_class_callback != NULL)
      {
          /* notify the application of the event */
          g_class_callback(controller_ID, event, val);
      }
}
```

4.2.4 Class request routine

This routine is called by USB Common Class Module. It handles class specific and vendor specific requests received from USB host. Vendor Specific requests are sent to Application using Vendor Specific application callback function already initialized with the class driver.

Pseudo Code:

```
static uint_8 USB_Class_XYZ_Requests (
      uint_8 controller_ID,            /* [IN] Controller ID */
      USB_SETUP_STRUCT * setup_packet, /*[IN] Setup packet */
      uint_8_ptr *data,           /* [OUT] Data to be send back */
      USB_PACKET_SIZE *size       /* [OUT] Size to be returned*/
)
{
      uint_8 index;
      uint_8 status = USBERR_INVALID_REQ_TYPE;
      uint_8 rpt_buf[REPORT_SIZE];/* buffer to send in case of get report req */
      *((uint_32_ptr)rpt_buf) = 0;

      if((setup_packet->request_type & USB_REQUEST_CLASS_MASK) ==
          USB_REQUEST_CLASS_CLASS)
      {
          /* class request so handle it here */
          <Class Specific Code goes here>
          if(g_param_callback != NULL)
          {
              /* notify the application of the class request.
                  Give control to the application */
              status = g_param_callback(setup_packet->request,
                  setup_packet->value,
                  data,
                  size);
          }
      }
      else if((setup_packet->request_type &
          USB_REQUEST_CLASS_MASK) ==
          USB_REQUEST_CLASS_VENDOR)
      {
          /* vendor specific request  */
          if(g_vendor_req_callback != NULL)
          {
              status = g_vendor_req_callback(controller_ID,
                  setup_packet,data, size);
          }
      }
      return status;
}
```

4.2.4.1 Endpoint service routine

This routine is called by USB Low Level Device Framework when data is sent or received on an endpoint. This routine is registered with the Low Level Device Framework by Class Driver during endpoint initialization.

Pseudo Code:

```
static void USB_Class_XYZ_Service_Endpoint (
      PTR_USB_EVENT_STRUCT event  /* [IN] Pointer to USB Event
                                        Structure */
)
{
      APP_DATA_STRUCT bulk_data;

      bulk_data.data_ptr = event->buffer_ptr;
      bulk_data.data_size = event->len;

      if(g_class_callback != NULL)
      {
          if(event->errors != 0)
          {
              <Class Specific Error Handling Code goes here>
              g_class_callback(event->controller_ID,
                  USB_APP_ERROR, (uint_8*)(&(event->errors)));
          }
          else
          {
              if(event->direction == USB_RECV)
              {
                  < Class Specific Data Receive Handling Code goes here>
                   g_class_callback(event->controller_ID,
                      USB_APP_DATA_RECEIVED,
                      (void*)&bulk_data);
              }
              else
              {
                  < Class Specific Data Send Complete Handling Code goes here>
                    g_class_callback(event->controller_ID,
                      USB_APP_DATA_SEND_COMPLETE,
                      (void*)&bulk_data);
              }
          }
      }
}
```

Chapter 5 Developing Applications

5.1 Introduction

This chapter discusses the functions used to develop applications based on the existing classes.

5.2 Application interfaces

The interfaces of the existing classes are defined keeping in mind that the application must be kept as independent as possible from the lower layer class drivers as well as drivers.

The interface definition between the application and classes is made up of the calls shown in Table 5-1.

Table 5-1. API calls

API Call	Description
Class Initialize	This API is used to initialize the class that in turn initializes not only the class but also initializes the lower driver layers.
Send Data	This API is used by the application to send the data to the host system. It is not recommended to make this call in a non-interrupt context.
Event Callback	All events on the bus are propagated to the application using the event callback. The data received on the bus is also propagated to the application using the event callback.
USB Vendor Specific Callback	This is an optional callback and is not mandatory for the application to support it. This callback is used to propagate any vendor specific request that the host system might have sent.
Periodic Task	This is an API call by the application to the class, so that it can complete some tasks that it may want to execute in non-interrupt context.

5.3 Developing an Application

Perform these steps to develop a new application:

1. Make a new application directory under /device/app directory. The new application directory is made to make the new application.
2. Copy the following files from the similar pre-existing applications.
   - main.c
   - usb_descriptor.c
   - usb_descriptor.h
   - user_config.h

Change these files to suit your application. The usb_descriptor.c and usb_descriptor.h files contain the descriptors for USB that are dependent on the application and the class driver. The user_config.h contains configuration options. The main.c file contains the initial code for the device. If the device is changed, this file must also be modified accordingly.

3. Create the CodeWarrior directory where the project files for the new application can be created.
4. Create a new file for creating the main application function and the callback function as defined above. In Figure 5-1, the new_app.c and new_app.h are used for the same purpose.

Figure 5-1. New application directory

- usb_descriptor.c

  This file contains USB Framework Module interface. It also contains various descriptors defined by USB Standards like, device descriptor, configuration descriptor, string descriptor and other class specific descriptors that are provided to Framework Module when requested. For customization, user can modify these variables and function implementations to suit the requirement.

  a) Variables

  The list below shows user modifiable variables for an already implemented class driver. The user should also modify corresponding MACROs defined in usb_descriptor.h file. For example, to save precious RAM space in S08 devices, constant variables are stored in ROM.

  – **usb_desc_ep**

  This is an array of endpoint structures. Endpoint structure describes the property of endpoint like, endpoint number, size, direction, type, and so on. This array should contain all the mandatory endpoints defined by USB class specifications.

  Sample code implementation of usb_desc_ep for HID class is given below:

```
const USB_ENDPOINTS usb_desc_ep =
{
        HID_DESC_ENDPOINT_COUNT,
        {
                HID_ENDPOINT,
                USB_INTERRUPT_PIPE,
                USB_SEND,
                HID_ENDPOINT_PACKET_SIZE,<---User Modifiable
        }
        < User can add other endpoints depending on class
        requirement >
};
```

– **g_device_descriptor**

This variable contains USB Device Descriptor.

Sample code implementation of device descriptor for HID class is given below:

```
uint_8 const g_device_descriptor[DEVICE_DESCRIPTOR_SIZE] =
{
        DEVICE_DESCRIPTOR_SIZE,         /* "Device Descriptor Size        */
        USB_DEVICE_DESCRIPTOR,          /* "Device" Type of descriptor    */
        0x00, 0x02,                     /*  BCD USB version               */
        0x00,                           /* Device Class is indicated in the
                                            interface descriptors */
        0x00,                           /* Device Subclass is indicated in
                                            the interface descriptors */
        0x00,                           /*  Device Protocol               */
        CONTROL_MAX_PACKET_SIZE,        /*  Max Packet size */
        0x04,0x25, <---User Modifiable /*  Vendor ID                     */
        0x00,0x01, <---User Modifiable /*  Product ID                    */
        0x02,0x00,                      /*  BCD Device version            */
        0x01,      <---User Modifiable /*  Manufacturer string index     */
        0x02,      <---User Modifiable /*  Product string index          */
        0x00,      <---User Modifiable /*  Serial number string index    */
        0x01                            /*  Number of configurations      */
};
```

– **g_config_descriptor**

This variable contains USB Configuration Descriptor.

Sample code implementation of configuration descriptor for HID class is given below:

```
uint_8 const g_config_descriptor[CONFIG_DESC_SIZE] =
{
        CONFIG_ONLY_DESC_SIZE,  /*  Configuration Descriptor Size - always 9
                                    bytes */
        USB_CONFIG_DESCRIPTOR,  /* "Configuration" type of descriptor */
        CONFIG_DESC_SIZE, 0x00, /*  Total length of the
                                    Configuration descriptor */
        1,                      /*  NumInterfaces */
        1,                      /*  Configuration Value */
        0,                      /*  Configuration Description String Index*/
        BUS_POWERED|SELF_POWERED|
                (REMOTE_WAKEUP_SUPPORT<<REMOTE_WAKEUP_SHIFT),
/* S08/CFV1/CFV2 are both self powered (its compulsory to set bus powered)*/
/*Attributes.support RemoteWakeup and self power*/
        0x32,      <---User Modifiable /*  Current draw from bus */

        /* Interface Descriptor */
        IFACE_ONLY_DESC_SIZE,
        USB_IFACE_DESCRIPTOR,
        0x00,
        0x00,
        HID_DESC_ENDPOINT_COUNT,
        0x03,
        0x01,
        0x02,
        0x00,

        /* HID descriptor */
        HID_ONLY_DESC_SIZE,
```

```
                USB_HID_DESCRIPTOR,
                0x00,0x01,
                0x00,
                0x01,
                0x22,
                0x34,0x00,

                /*Endpoint descriptor */
                ENDP_ONLY_DESC_SIZE,
                USB_ENDPOINT_DESCRIPTOR,
                HID_ENDPOINT|(USB_SEND << 7),
                USB_INTERRUPT_PIPE,
                HID_ENDPOINT_PACKET_SIZE, 0x00, <---User Modifiable
                0x0A
};
```

– **String Descriptors**

Users can modify string descriptors to customize their product. String descriptors are written in UNICODE format. An appropriate language identification number is specified in USB_STR_0. Multiple language support can also be added.

Sample code implementation of string descriptors for HID class application is given below:

```
uint_8 const USB_STR_0[USB_STR_0_SIZE+USB_STR_DESC_SIZE] =
{
          sizeof(USB_STR_0),
          USB_STRING_DESCRIPTOR,
          0x09,
          0x04/*equiavlent to 0x0409*/
          <User can add other language support descriptor here>
};

uint_8 const USB_STR_1[USB_STR_1_SIZE+USB_STR_DESC_SIZE] =
{
          sizeof(USB_STR_1),
          USB_STRING_DESCRIPTOR,
          <User Modifiable Manufacturer Name in UNICODE>
          'F',0,
          'R',0,
          'E',0,
          'E',0,
          'S',0,
          'C',0,
          'A',0,
          'L',0,
          'E',0,
          ' ',0,
          'S',0,
          'E',0,
          'M',0,
          'I',0,
          'C',0,
          'O',0,
          'N',0,
          'D',0,
          'U',0,
          'C',0,
          'T',0,
```

```
        'O',0,
        'R',0,
        ' ',0,
        'I',0,
        'N',0,
        'C',0,
        '.',0
};

uint_8 const USB_STR_2[USB_STR_2_SIZE+USB_STR_DESC_SIZE] =
{
        sizeof(USB_STR_2),
        USB_STRING_DESCRIPTOR,
        <User Modifiable Product Name in UNICODE>
        ' ',0,
        ' ',0,
        'J',0,
        'M',0,
        ' ',0,
        'H',0,
        'I',0,
        'D',0,
        ' ',0,
        'D',0,
        'E',0,
        'M',0,
        'O',0,
        ' ',0
};

uint_8 const USB_STR_n[USB_STR_n_SIZE+USB_STR_DESC_SIZE] =
{
        sizeof(USB_STR_n),
        USB_STRING_DESCRIPTOR,
        'B',0,
        'A',0,
        'D',0,
        ' ',0,
        'S',0,
        'T',0,
        'R',0,
        'I',0,
        'N',0,
        'G',0,
        ' ',0,
        'I',0,
        'N',0,
        'D',0,
        'E',0,
        'X',0
};

uint_8 const g_string_desc_size[USB_MAX_STRING_DESCRIPTORS+1] =
{
        sizeof(USB_STR_0),
        sizeof(USB_STR_1),
        sizeof(USB_STR_2),
```

```
        <User can add other string descriptors sizes here>
        sizeof(USB_STR_n)
};

uint_8_ptr const g_string_descriptors[USB_MAX_STRING_DESCRIPTORS+1] =
{
        (uint_8_ptr const) USB_STR_0,
        (uint_8_ptr const) USB_STR_1,
        (uint_8_ptr const) USB_STR_2,
        <User can add other string descriptors here>
        (uint_8_ptr const) USB_STR_n
};

USB_ALL_LANGUAGES g_languages =
{
        USB_STR_0, sizeof(USB_STR_0),
        {
                (uint_16 const)0x0409,
                (const uint_8 **)g_string_descriptors,
                g_string_desc_size
        }
        <User can add other language string descriptors here>
};
```

– **Standard Descriptor Table**

Users can modify standard descriptor table to support additional class specific descriptors and vendor specific descriptors.

Sample implementation below is shown for HID Class application.

```
USB_PACKET_SIZE const g_std_desc_size[USB_MAX_STD_DESCRIPTORS+1] =
{
        0,
        DEVICE_DESCRIPTOR_SIZE,
        CONFIG_DESC_SIZE,
        0, /* string */
        0, /* Interface */
        0, /* Endpoint */
        0, /* Device Qualifier */
        0, /* other speed config */
        <Other Descriptor Sizes goes here>
        REPORT_DESC_SIZE
};

uint_8_ptr const g_std_descriptors[USB_MAX_STD_DESCRIPTORS+1] =
{
        NULL,
        (uint_8_ptr)g_device_descriptor,
        (uint_8_ptr)g_config_descriptor,
        NULL, /* string */
        NULL, /* Interface */
        NULL, /* Endpoint */
        NULL, /* Device Qualifier */
        NULL, /* other speed config*/
        <Other Descriptor pointers go here>
        Sample HID Class Report Desc->
                (uint_8_ptr)g_report_descriptor
};
```

– **g_valid_config_values**

This variable contains valid configurations for a device. This value remains fixed for a device.

```
uint_8 const g_valid_config_values[USB_MAX_CONFIG_SUPPORTED+1]={0,1};
```

– **g_alternate_interface**

This variable contains valid alternate interfaces for a given configuration. Sample implementation uses a single configuration. If user is implementing additional alternate interfaces then USB_MAX_SUPPORTED_INTERFACES macro (usb_descriptor.h) should be changed accordingly.

```
static uint_8 g_alternate_interface[USB_MAX_SUPPORTED_INTERFACES];
```

b) Interfaces

The following interfaces are required to be implemented by Application in usb_descriptor.c. These interfaces are called by low level USB stack and class drivers. Refer to *Freescale USB Stack Device API Reference Manual* (document *USBAPIRM.pdf*) for details regarding interface functions along with sample implementation. Also, refer to usb_descriptor.c and usb_descriptor.h files in device\app\hid for reference.

– **USB_Desc_Get_Descriptor**

This interface function is invoked by USB Framework. This call is made when Framework receives GET_DESCRIPTOR call from Host. Mandatory descriptors that an application is required to implement are:

– Device Descriptor
– Configuration Descriptor
– Class Specific Descriptors (For example, for HID class implementation, Report Descriptor and HID Descriptor)

Apart from the mandatory descriptors, an application should also implement various string descriptors as specified by the Device Descriptor and other configuration descriptors.

Sample code for HID class application is given below.

```
uint_8 USB_Desc_Get_Descriptor(
     uint_8 controller_ID,   /* [IN]  Controller ID */
     uint_8 type,            /* [IN]  Type of descriptor requested */
     uint_8 str_num,     /* [IN]  String index for string descriptor */
     uint_16 index,          /* [IN]  String descriptor language Id */
     uint_8_ptr *descriptor, /* [OUT] Output descriptor pointer */
     USB_PACKET_SIZE *size   /* [OUT] Size of descriptor returned */
)
{
    #pragma unused (controller_ID)
    switch(type)
    {
      <Class Specific Descriptor code goes here >
      case USB_REPORT_DESCRIPTOR:
        {
          type = USB_MAX_STD_DESCRIPTORS;
          *descriptor = (uint_8_ptr)g_std_descriptors [type];
          *size = g_std_desc_size[type];
```

```
    }
    break;
  case USB_HID_DESCRIPTOR:
    {
      type = USB_CONFIG_DESCRIPTOR ;
      *descriptor = (uint_8_ptr)(g_std_descriptors [type]+
           CONFIG_ONLY_DESC_SIZE+IFACE_ONLY_DESC_SIZE);
      *size = HID_ONLY_DESC_SIZE;
    }
    break;
  case USB_STRING_DESCRIPTOR:
    {
        if(index == 0)
        {
            /* return the string and size of all languages */
            *descriptor = (uint_8_ptr)g_languages.
                languages_supported_string;
            *size = g_languages.languages_supported_size;
        }
        else
        {
            uint_8 lang_id = 0;
            uint_8 lang_index = USB_MAX_LANGUAGES_SUPPORTED;

            for(;lang_id < USB_MAX_LANGUAGES_SUPPORTED; lang_id++)
            {
                /* check whether we have a string for this language
                */
                if(index ==
                    g_languages.usb_language[lang_id].language_id)
                {
                    /* check for max descriptors */
                    if(str_num < USB_MAX_STRING_DESCRIPTORS)
                    {
                        /* setup index for the string to be
                           returned */
                        lang_index = str_num;
                    }
                    break;
                }
            }

            /* set return val for descriptor and size */
            *descriptor =
                (uint_8_ptr)g_languages.usb_language[lang_id].
                lang_desc[lang_index];
            *size = g_languages.usb_language[lang_id].
                lang_desc_size[lang_index];
        }
    }
    break;
  default :
    if (type < USB_MAX_STD_DESCRIPTORS)
    {
        /* set return val for descriptor and size*/
        *descriptor = (uint_8_ptr)g_std_descriptors [type];
```

```
            /* if there is no descriptor then return error */
            if(*descriptor == NULL)
            {
                return USBERR_INVALID_REQ_TYPE;
            }

            *size = g_std_desc_size[type];
        }
        else /* invalid descriptor */
        {
            return USBERR_INVALID_REQ_TYPE;
        }
        break;
    }
    return USB_OK;
}
```

– **USB_Desc_Get_Endpoints**

This interface function is called from class driver. This function returns a pointer to USB_ENDPOINT structure. This structure describes the characteristics of Non Control Endpoint, for example. endpoint number, type, direction, and size or any other endpoint characteristic required by class driver implementation.

Sample implementation is given below.

```
void* USB_Desc_Get_Endpoints
(
        uint_8 controller_ID      /* [IN] Controller ID */
)
{
        #pragma unused (controller_ID)
        return (void*)&usb_desc_ep;
}
```

– **USB_Desc_Get_Interface**

This interface function invoked by USB Framework. This function returns a pointer to alternate interface for the specified interface. This routine is called when USB Framework receives GET_INTERFACE request from Host.

Sample code for single configuration HID class is given below.

```
uint_8 USB_Desc_Get_Interface
(
        uint_8 controller_ID,     /* [IN] Controller ID */
        uint_8 interface,         /* [IN] Interface number */
        uint_8_ptr alt_interface  /* [OUT] Output alternate interface */
)
{
        #pragma unused (controller_ID)
        /* if interface valid */
        if(interface < USB_MAX_SUPPORTED_INTERFACES)
        {
                <User can modify this to support multiple configurations>
                /* get alternate interface*/
                *alt_interface = g_alternate_interface[interface];
                return USB_OK;
        }
```

```
        return USBERR_INVALID_REQ_TYPE;
}
```

– **USB_Desc_Remote_Wakeup**

This interface function invoked by USB Framework. If the application supports remote wakeup then this function return TRUE otherwise FALSE. If the application supports remote wakeup, then USB Device Descriptor should support this capability. If user does not support remote wakeup, then set REMOTE_WAKEUP_SUPPORT should be set to 0 in usb_descriptor.h.

Sample code is given below.

```
boolean USB_Desc_Remote_Wakeup
(
        uint_8 controller_ID      /* [IN] Controller ID */
)
{
        #pragma unused (controller_ID)
        return REMOTE_WAKEUP_SUPPORT;
}
```

– **USB_Desc_Set_Interface**

This interface function is called from USB Framework. This function sets an alternate interface for specified interface. This routine is called when USB Framework receives SET_INTERFACE request from the host.

Sample code for single configuration HID class is given below.

```
uint_8 USB_Desc_Set_Interface
(
        uint_8 controller_ID, /* [IN] Controller ID */
        uint_8 interface,     /* [IN] Interface number */
        uint_8 alt_interface  /* [IN] Input alternate interface */
)
{
        #pragma unused (controller_ID)
        /* if interface valid */
        if(interface < USB_MAX_SUPPORTED_INTERFACES)
        {
                <User can modify this to support multiple configurations>
                /* set alternate interface*/
                g_alternate_interface[interface] = alt_interface;
                return USB_OK;
        }

        return USBERR_INVALID_REQ_TYPE;
}
```

– **USB_Desc_Valid_Configation**

This interface function is called from USB Framework. This function returns if the configuration is valid or not. This routine is called when USB Framework receives SET_CONFIGURATION request from the host.

Sample code for single configuration HID class is given below.

```
boolean USB_Desc_Valid_Configation(
(
        uint_8 controller_ID, /*[IN] Controller ID */
        uint_16 config_val   /*[IN] Configuration value */
)
{
        #pragma unused (controller_ID)
        uint_8 loop_index=0;
        /* check with only supported val right now */
        while(loop_index < (USB_MAX_CONFIG_SUPPORTED+1))
        {
                if(config_val == g_valid_config_values[loop_index])
                {
                        return TRUE;
                }
                loop_index++;
        }
        return FALSE;
}
```

– **USB_Desc_Valid_Interface**

This interface function is called from class driver to validate if the interface is valid or not. This function returns TRUE if interface is valid, otherwise FALSE.

Sample code for single configuration HID class is given below.

```
boolean USB_Desc_Valid_Interface
(
        uint_8 controller_ID, /*[IN] Controller ID */
        uint_8 interface      /*[IN] Target interface */
)
{
        #pragma unused (controller_ID)
        uint_8 loop_index=0;
        <User can modify this to support multiple configurations>
        /* check with only supported val right now */
        while(loop_index < USB_MAX_SUPPORTED_INTERFACES)
        {
                if(interface == g_alternate_interface[loop_index])
                {
                        return TRUE;
                }
                loop_index++;
        }
        return FALSE;
}
```

Apart from above interfaces mandated by USB Low Level Framework, the application must also implement various callback functions to receive events from USB class driver. These interface functions are implemented in new_app.c file.

— Class Callback function

This function is used by class driver to inform application about various USB Bus Events. The application can use these events to perform event specific functionalities. The implementation of this callback function is governed by class driver specification.

Pseudo Code:

```
void USB_App_Class_Callback
(
        uint_8 controller_ID,/* [IN] Controller ID */
        uint_8 event_type,   /* [IN] value of the event*/
        void* val     /* [IN] gives the configuration value*/
)
{
        if(event_type == USB_APP_BUS_RESET)
        {
                /* USB Bus Reset */
                <Application Specific Code goes here>
        }
        else if(event_type == USB_APP_ENUM_COMPLETE)
        {
                /* Enumeration is complete */
                <Application Specific Code goes here>
        }
        return;
}
```

— Vendor Request Callback

This optional argument allows application to support any vendor specific USB requests received from USB host. This function allows application developer to enhance existing class implementation by adding vendor specific functionality.

Pseudo Code:

```
uint_8 USB_App_Vendor_Request_Callback
(
        uint_8 request,      /* [IN] request type */
        /* [IN] Pointer to Setup Packet */
        USB_SETUP_STRUCT *setup,
        uint_8_ptr* data,    /* [OUT] pointer to the data */
        USB_PACKET_SIZE* size/* [OUT] size of the transfer */
)
{
        uint_8 status = USB_OK;
        uint_8 request = setup-> request;
        switch(request)
        {
                < Vendor Specific Requests are handled here >
                case <VENDOR_REQUEST1> :
                        *data = <Pointer to Data to be sent>
                        *size = <size of Data to be sent>
                        break;

                case <VENDOR_REQUEST2>:
                        *data = <Pointer to Data to be sent>
                        *size = <size of Data to be sent>
                        break;
                        :
                        :
                        :
                default:
                        < UNHANDLED Vendor Specific Requests
                        Application code goes here>
                        status = USBERR_INVALID_REQ_TYPE;
```

```
                        break;
                }
                return status;
        }
```

— Class Specific Request Callback

Implementation of this function is governed by Class Driver Design. The design and implementation details are however beyond the scope.

Pseudo Code:

```
uint_8 USB_App_Class_Spec_Request_Callback
(
        uint_8 request,      /* [IN] request type */
        uint_16 value,       /* [IN] report type and ID */
        uint_8_ptr* data,    /* [OUT] pointer to the data */
        USB_PACKET_SIZE* size/* [OUT] size of the transfer */
)
{
        uint_8 status = USB_OK;
         *size = 0;
        /* handle the class request */
        switch(request)
        {
                < Class Specific Requests are handled here >
                case <REQUEST1>:
                        *data = <Pointer to Data to be sent>
                        *size = <size of Data to be sent>
                        break;

                case <REQUEST2>:
                        *data = <Pointer to Data to be sent>
                        *size = <size of Data to be sent>
                        break;
                        :
                        :
                        :
                default:
                        < UNHANDLED Class Specific Requests
                        Application code goes here>
                        status = USBERR_INVALID_REQ_TYPE;
                        break;
        }
        return status;
}
```

- usb_descriptor.h

  This file is mandatory for the application to implement. Framework and class drivers include this file for function prototype definitions and data structures described in usb_descriptor.c. User modifying usb_descriptor.c should also modify MACROs in this file as well.

- user_config.h

  This file is required to define various compile time macros. These parameters are essential for successful compilation of source code.

  Madatory macros that need to be defined are:

- LONG_SEND_TRANSACTION — This macro is defined for classes that have to send data more than the endpoint size over USB bus. It allows Low Level Device Framework to use split transaction for sending data over USB bus.
- LONG_RECIEVE_TRANSACTION — This macro is defined for classes that have to receive data more than the endpoint size over USB bus. It allows Low Level Device Framework to use split transaction for receiving data over USB bus.
- USB_PACKET_SIZE — This macro defines maximum transfer packet size of USB Data Transaction.
- MAX_TIMER_OBJECTS — This macro defines number of timer objects required by application.
- TIMER_CALLBACK_ARG — This macro if defined invokes timer callbacks with an application specified argument.
- DOUBLE_BUFFERING_USED — For S08 devices, if application is making use of endpoint 5 and 6 (Double Buffered) then this macro has to be defined. For CFV1 and CFV2 devices, this macro is not required, as all the endpoints are double buffered.

5. After the functionality has been implemented, use the steps defined in A.1.2, “Building the Application with CodeWarrior 6 and CodeWarrior 7” and A.1.3, “Running the Application with CodeWarrior 6 and CodeWarrior 7” to build and run the application.

5.4 Application design

This section discusses the application design. The application is made up of the main application function and the callback function.

5.4.1 Main Application Function

The main application function uses the following C code:

```
void TestApp_Init(void)
{
    uint_8 error;
    DisableInterrupts;

    <Application Specific Initialization Code goes here>
    /* Initialize the USB Class Driver interface */
    <USB Class Initialization Call>
    error =
        USB_Class_XYZ_Init(0, USB_App_Callback,
        NULL, USB_App_Param_Callback);

    EnableInterrupts;
    while (TRUE)
    {
        __RESET_WATCHDOG();
        <Application Specific Code goes here>
          new_app_task();
    }/* Endwhile */
}
```

5.4.2 Callback Function

The callback function uses the following C code:

```
void USB_App_Callback(uint_8 controller_ID, uint_8 event_type, void* val)
{
    if(event_type == USB_APP_BUS_RESET)
    {
        <Application Specific Code goes here>
    }
    else if(event_type == USB_APP_ENUM_COMPLETE)
    {
         /* if enumeration is complete */
        <Application Specific Code goes here>
    }
    else if(event_type == USB_APP_ENUM_ERROR)
    {
        <Application Specific Code goes here>
    }
    return;
}
```

Appendix A Working with the Software

A.1 Introduction

This chapter gives you insight on how to use the Freescale USB Stack software. The following sections are described in this chapter:

- Preparing the setup
- Building the application
- Running the application

Knowledge of CodeWarrior IDE will be helpful to understand this section. While reading this chapter, practice the steps mentioned.

To take you through this chapter, the HID mouse application for the MC9S08JM60 is used as an example. For preparing the setup, building the application, and running the application the following devices—Kinetis, HC(S)08, ColdFire v1, and ColdFire V2—are used as an example.

A.1.1 Preparing the setup

A.1.1.1 Software setup

1. Double-click the Freescale_USB_Stack_v[*current version*].exe installer executable file.
2. The Freescale USB Stack Setup window appears. The following example shows the demonstration for USB Stack installation. You can follow the same instructions for new versions.

Example:

1. Click on the **Next** button to continue with Freescale USB Stack Setup installation.

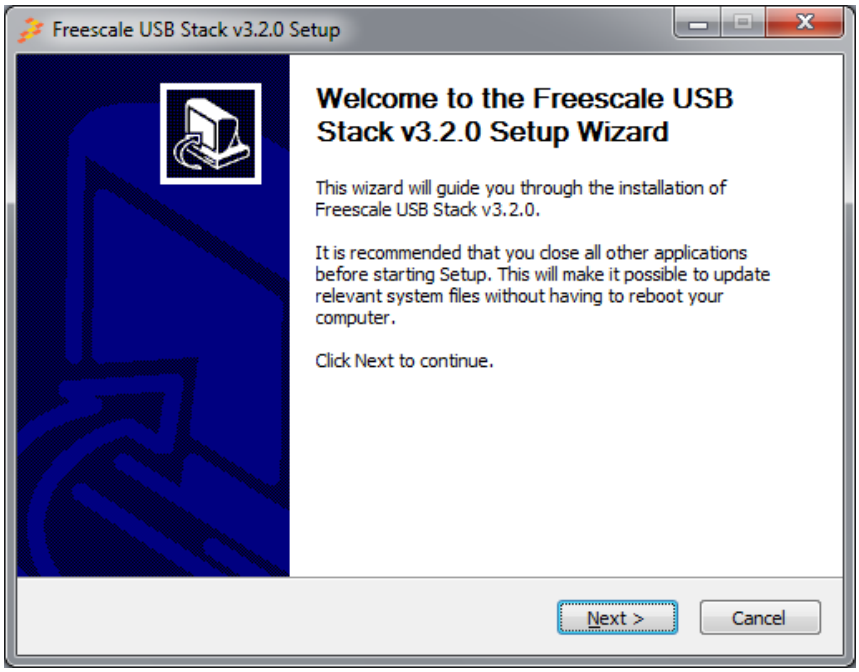


Figure A-1. Freescale USB Stack setup wizard

2. In Figure A-2, click on the **I Agree** button to accept the license agreement.

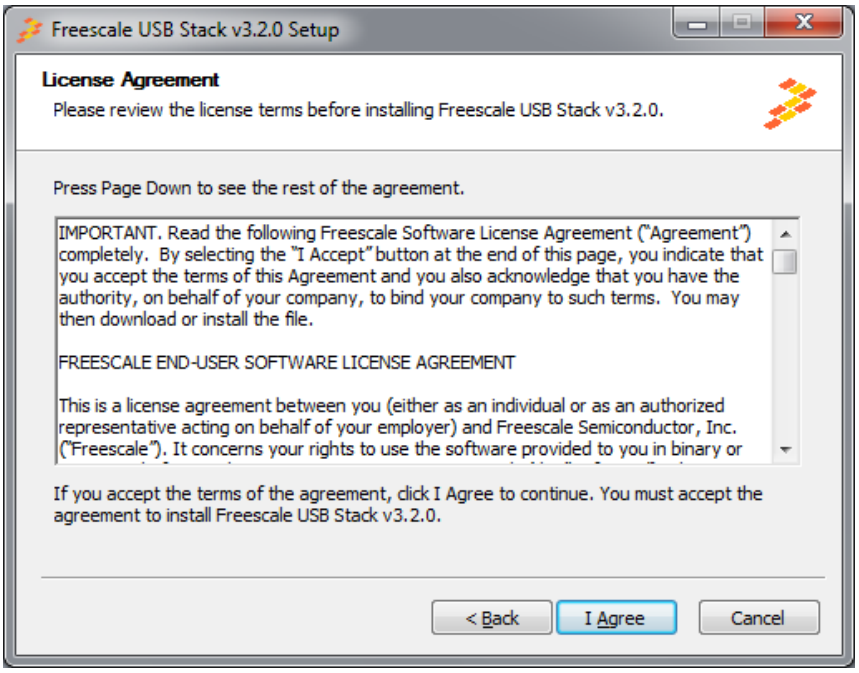


Figure A-2. Freescale USB Stack setup license agreement

3. In Figure A-3, select USB low level stack and other class components to install and click on the **Next** button.

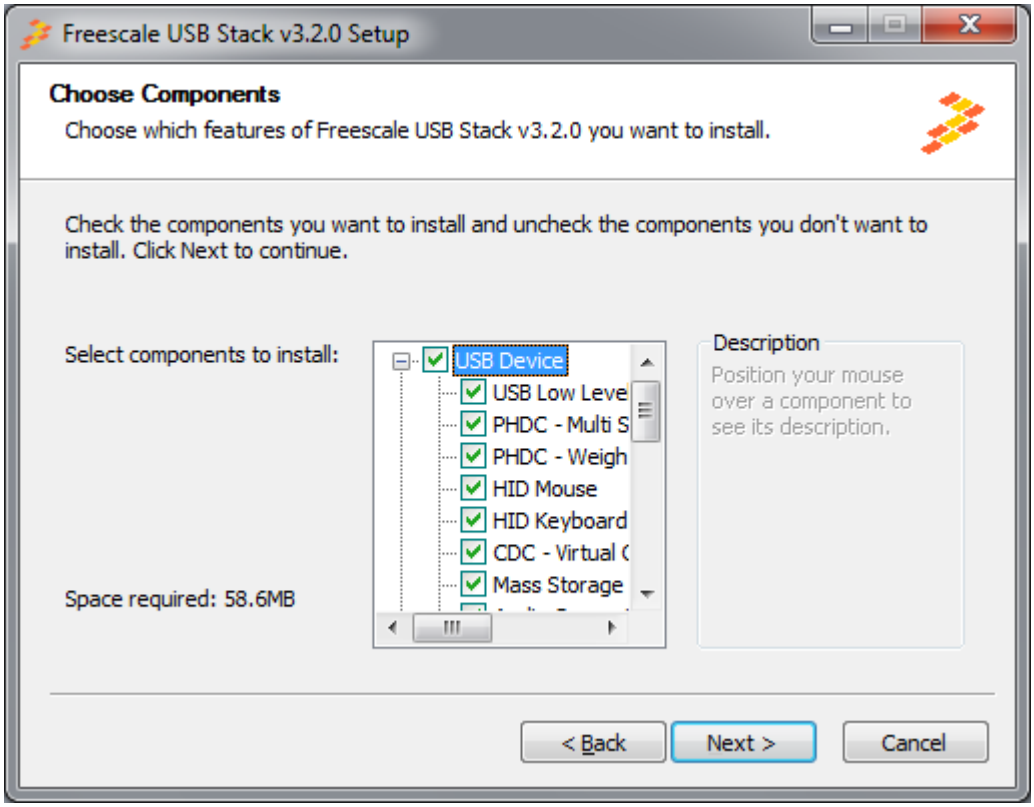


Figure A-3. Freescale USB Stack components

4. In Figure A-4, select the location of the folder where you require to install the Freescale USB Stack software and click on the **Install** button.

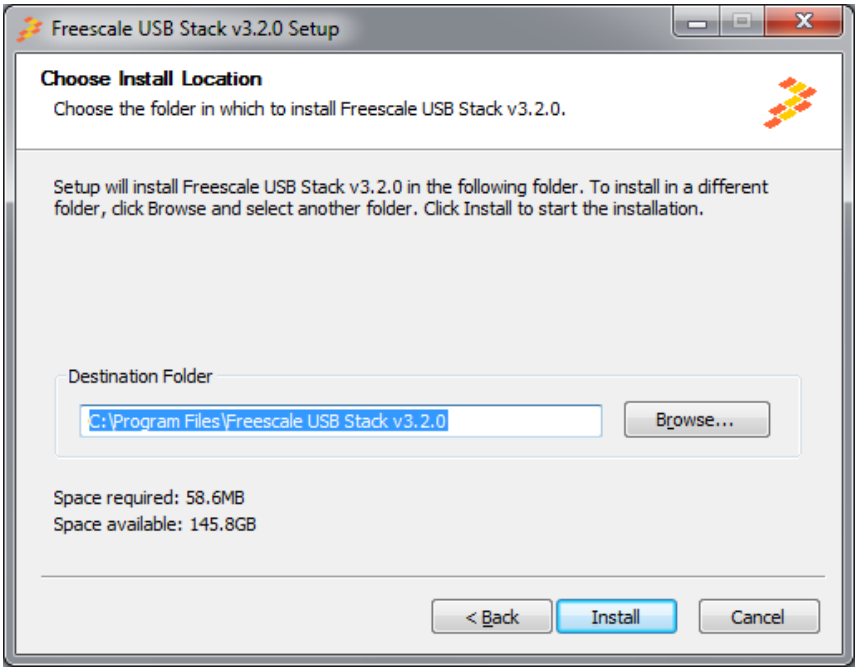


Figure A-4. Freescale USB Stack installation folder location

5. Click on the **Finish** button to successfully complete the Freescale USB Stack Setup Wizard.

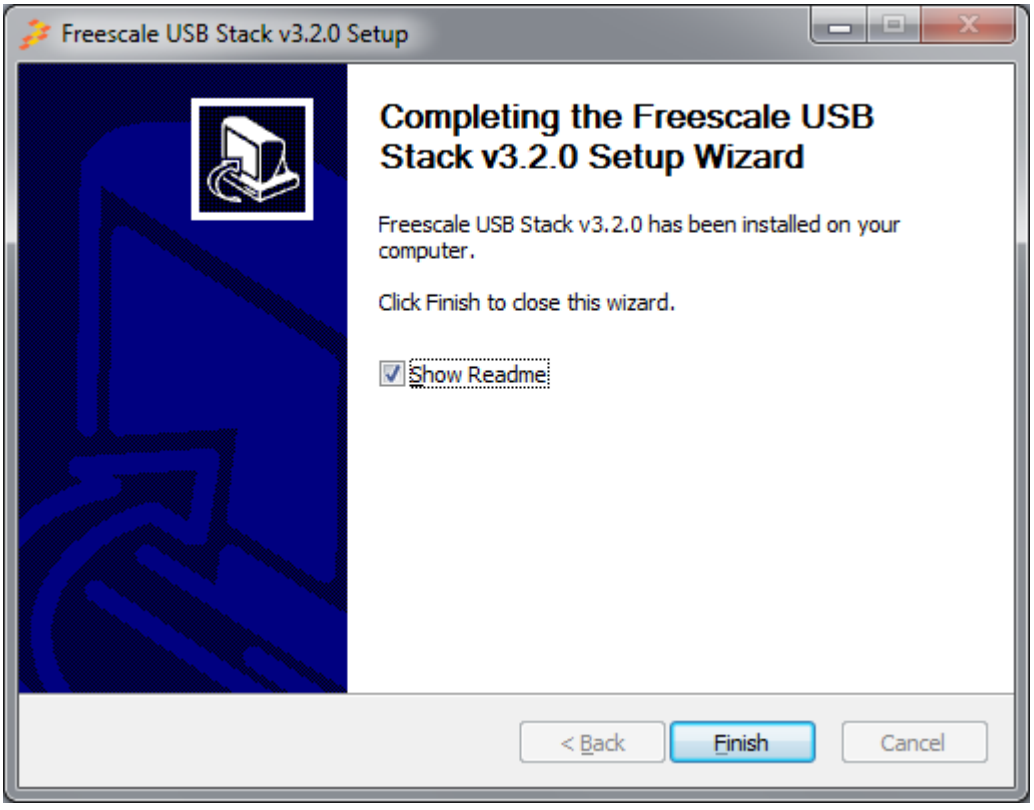


Figure A-5. Freescale USB Stack installation finish

Launching Freescale USB Stack project

Click **Start** > **Programs** > **Freescale USB Stack** > **Source** to launch the project.

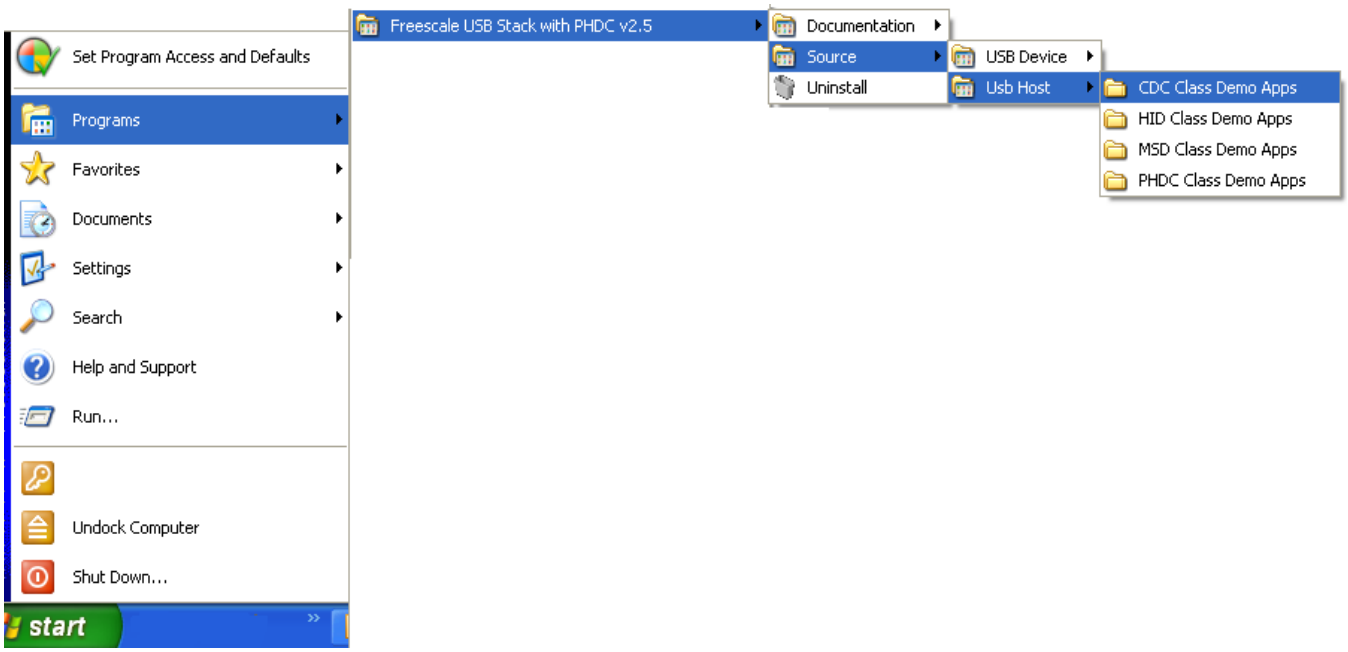


Figure A-6. Freescale USB Stack source program for launch

A.1.1.2 Hardware setup

- Make the connections as shown in Figure A-7. Here for hardware setup S08 is used as an example.

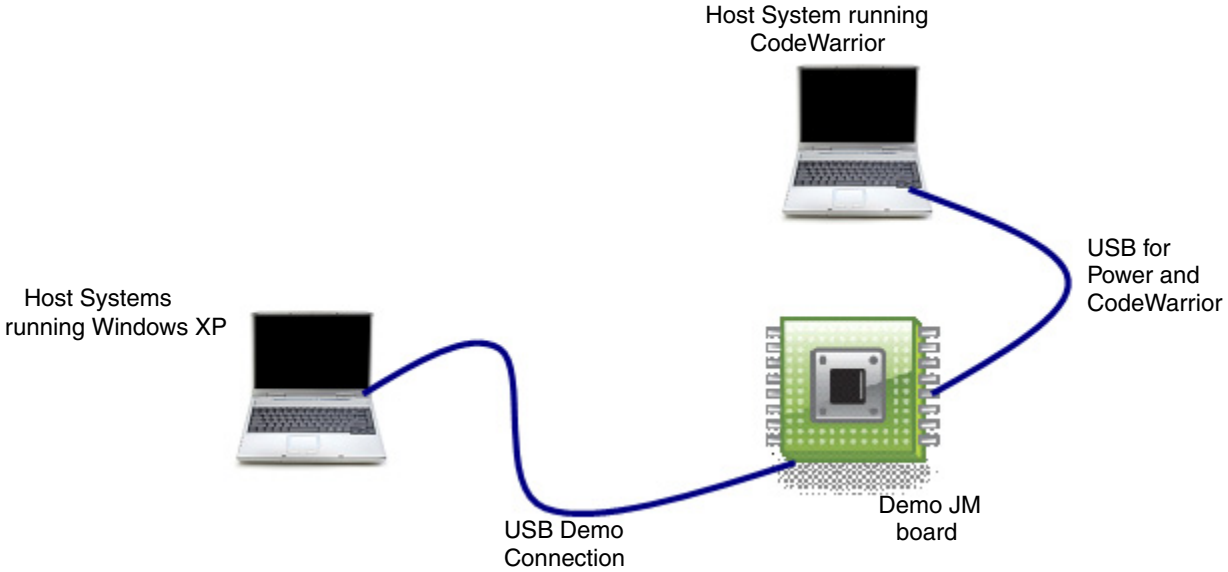


Figure A-7. S08 USB setup

— Make the first USB connection between the personal computer where the software is installed and the DemoJM board where the silicon is mounted. This connection is required to provide power to the board and downloading image to the flash.

— Make the second connection between the DemoJM board and the personal computer where the demo is run.

NOTE

Although, we have used two personal computers in Figure A-7, in reality you may achieve the same result by a single personal computer with two or more USB ports.

A.1.2 Building the Application with CodeWarrior 6 and CodeWarrior 7

The software for S08 and CFV1 is built with CodeWarrior 6.3. In addition, the software for CFV2 is built with CodeWarrior 7.2. Therefore, it contains application project files that can be used to build the project.

Before starting the process of building the project, make sure CodeWarrior 6.3 is installed on your computer.

To build the S08 project:

1. Navigate to the project file and open the s08usbjm60.mcp project file in CodeWarrior IDE.

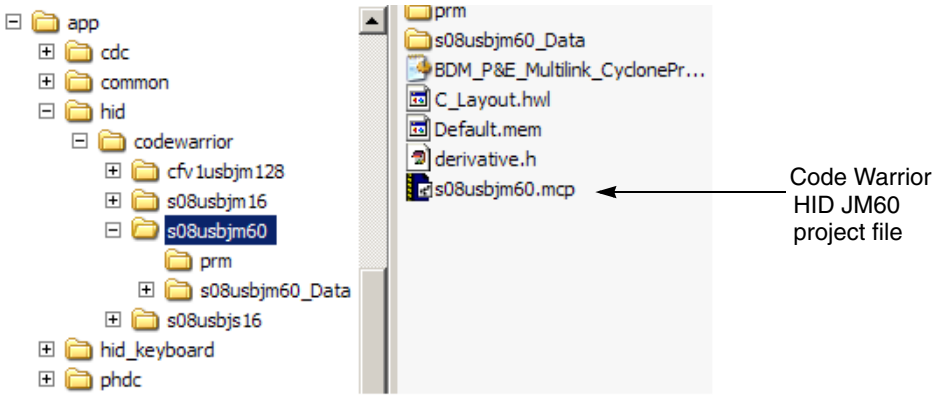


Figure A-8. Open s08usbjm60.mcp project file

2. After you have opened the project, the following window appears. To build the project, click the button as shown in Figure A-9.

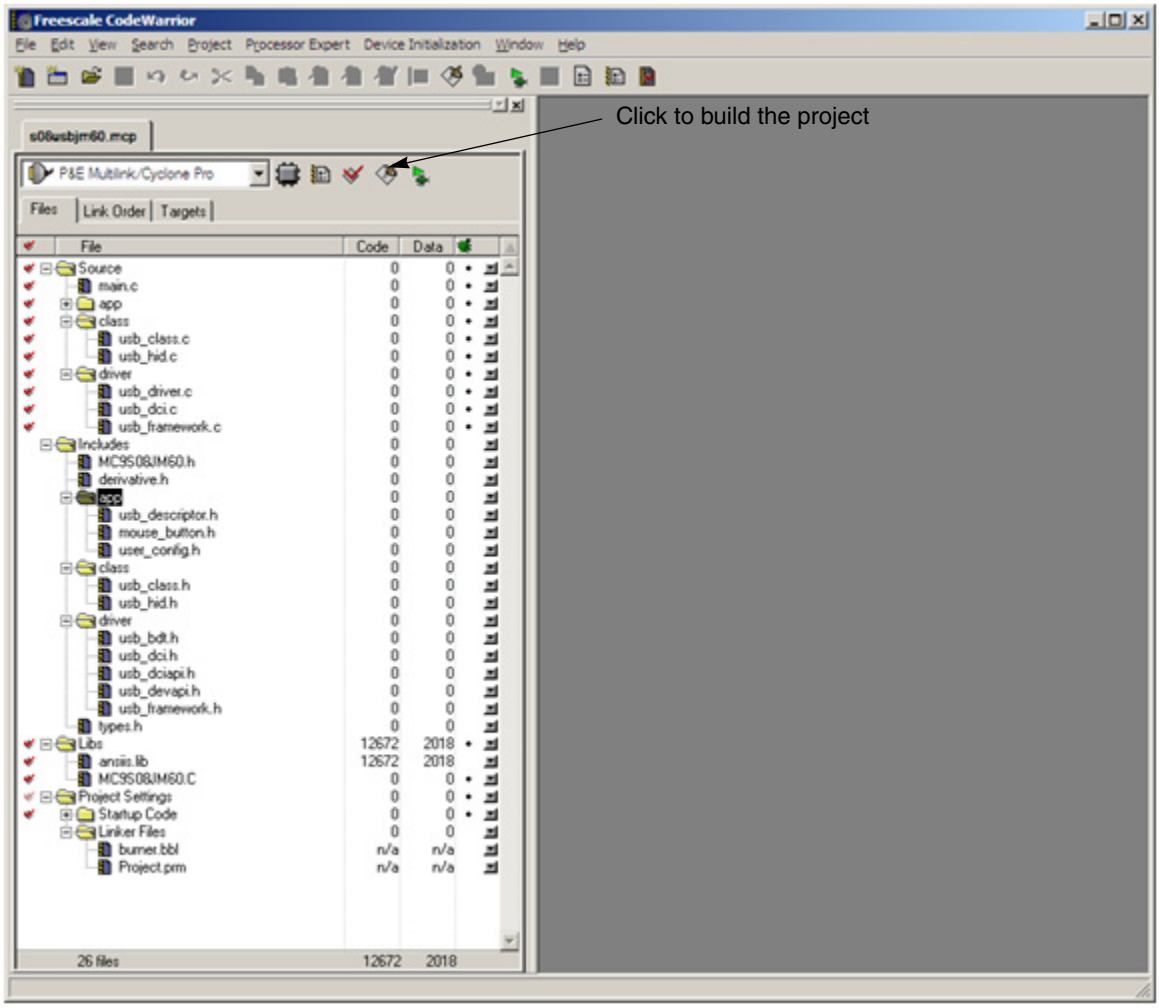


Figure A-9. Build s08usbjm60.mcp project

3. After the project is built, the code and data columns must appear filled across the files.

NOTE

The above procedure can be used to build CFV1 and CFV2 projects also.

A.1.3 Running the Application with CodeWarrior 6 and CodeWarrior 7

Refer to the board documentation and CodeWarrior manual for details on how to program the flash memory on the evaluation board used. The following steps are presented as an example about how to run the HID mouse application with DemoJM60 board using a P&E-micro debugger.

1. To run the application, click the button as shown in Figure A-10.

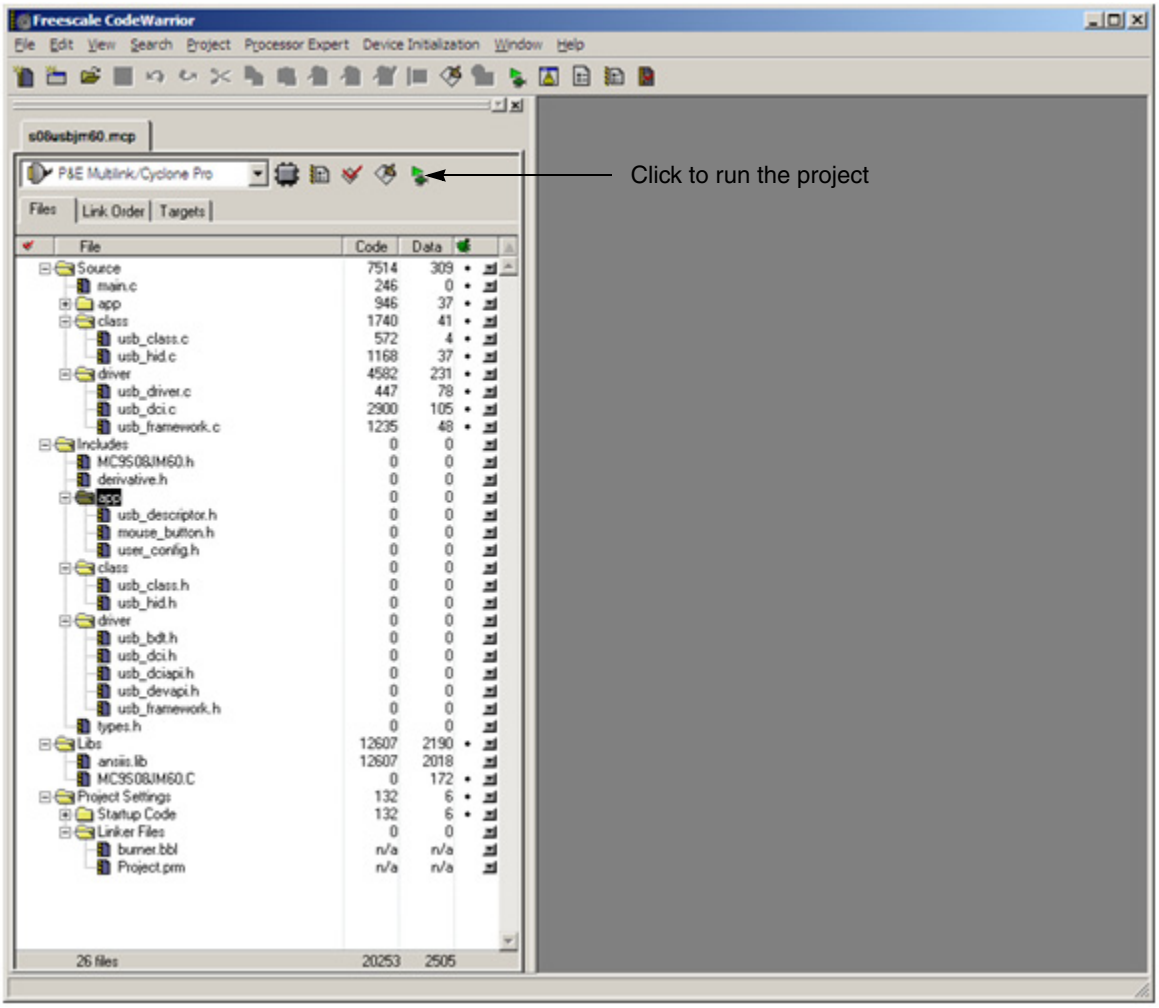


Figure A-10. Running the application

2. The dialog box in Figure A-11 appears. Click on the **Connect (Reset)** button to connect to hardware as shown in Figure A-11.

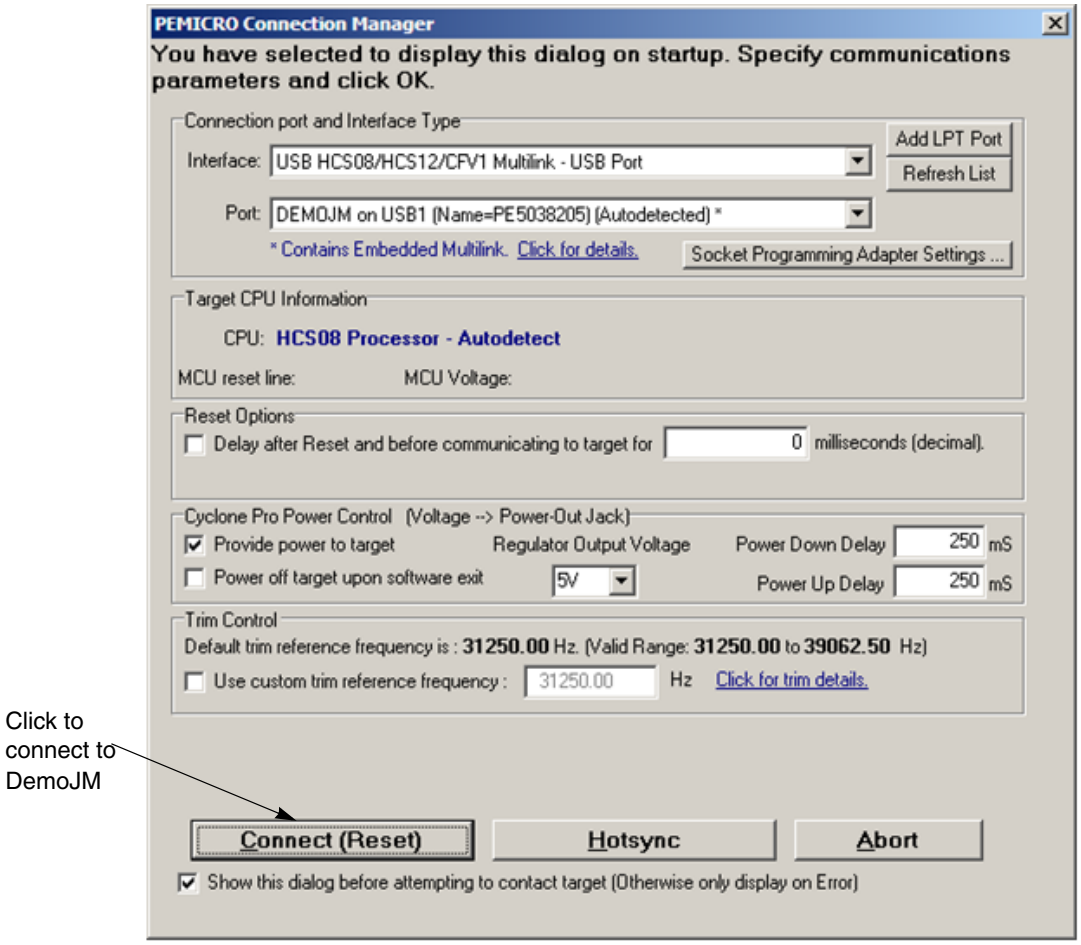


Figure A-11. Connection Manager

3. The pop-up in Figure A-12 appears. Click on the **Yes** button to load the built image to the JM60 flash.

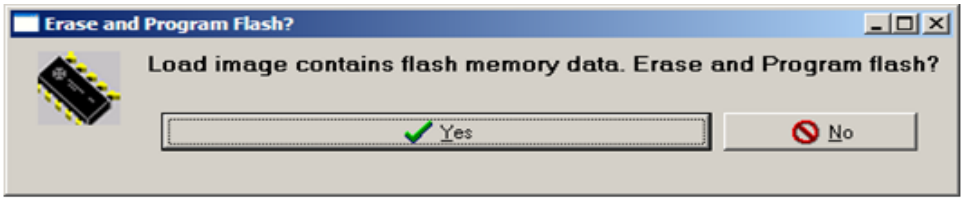


Figure A-12. Erase and Program Flash pop-up

4. The pop-up in Figure A-13 appears to erase and program the built image to the JM60 flash.

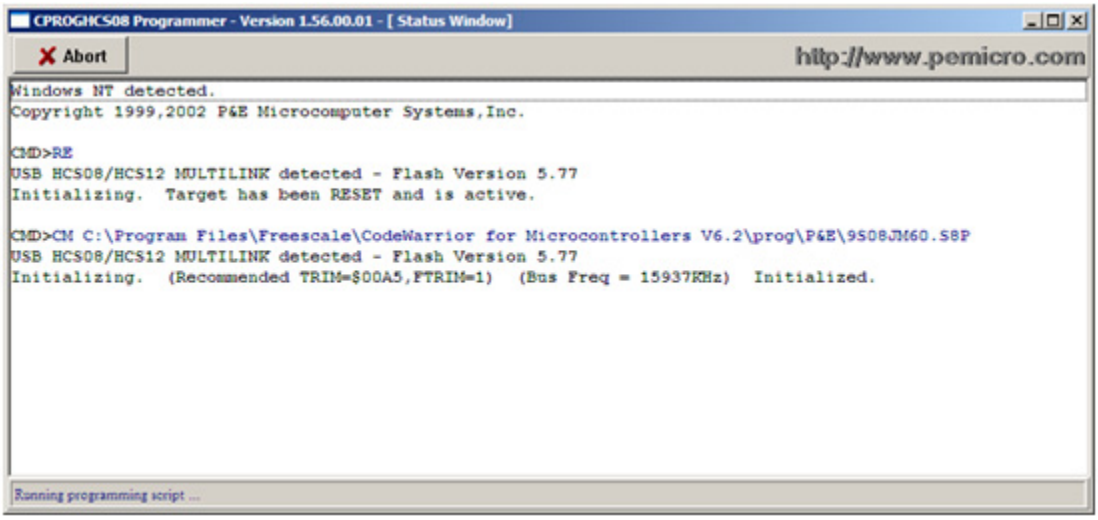


Figure A-13. Image Programmed in Flash

5. After the image is programmed in the flash, the debugger window as shown in Figure A-14 appears. Click on the Green Arrow as shown in Figure A-14 to run the programmed image.

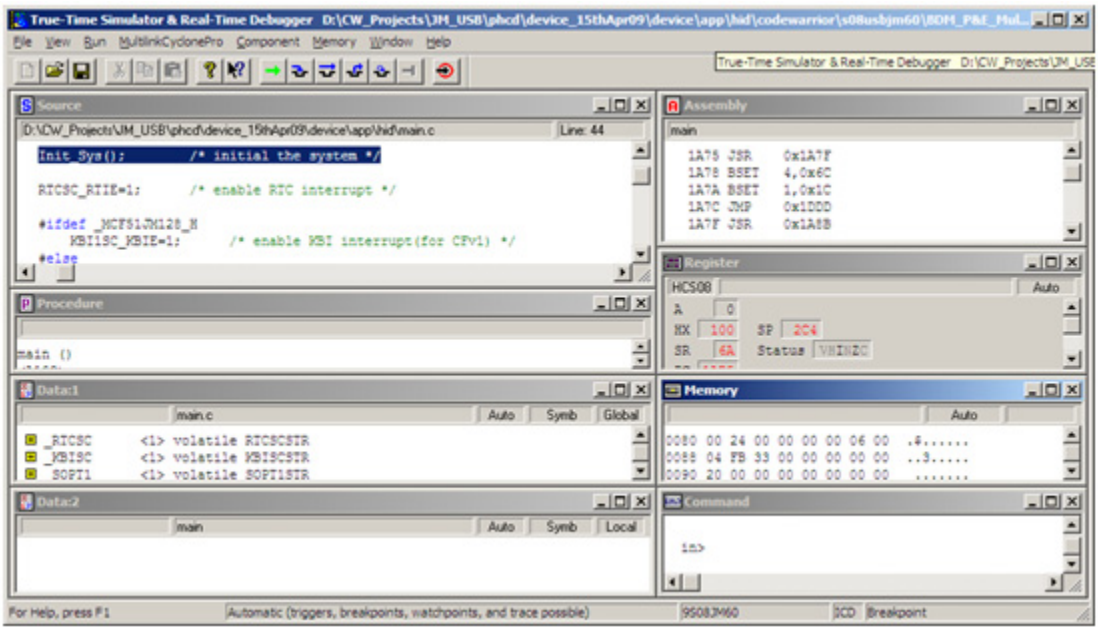


Figure A-14. Simulator and Real-Time Debugger

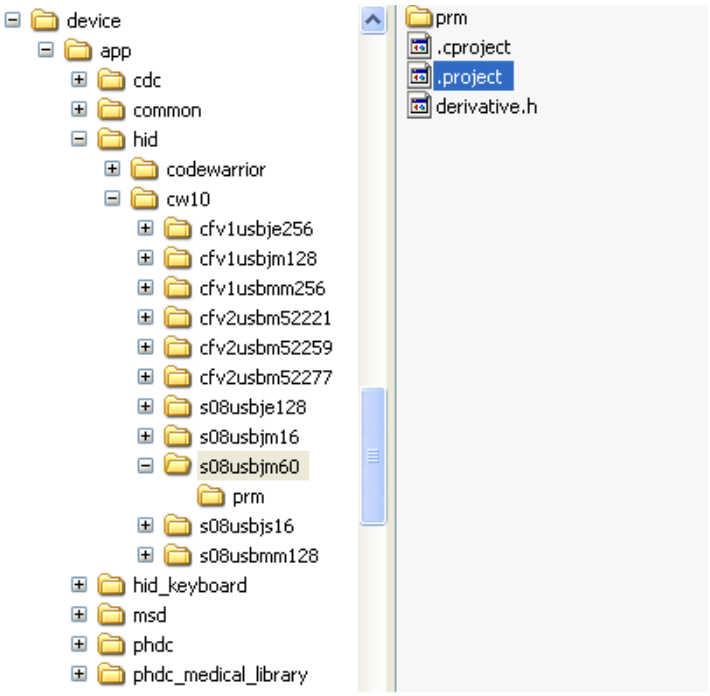
A.1.4 Building and Running the Application with CodeWarrior 10

The software for Kinetis k40, S08, CFV1 and CFV2 targets is available to be build, download and debug using the CodeWarrior 10 MCU.

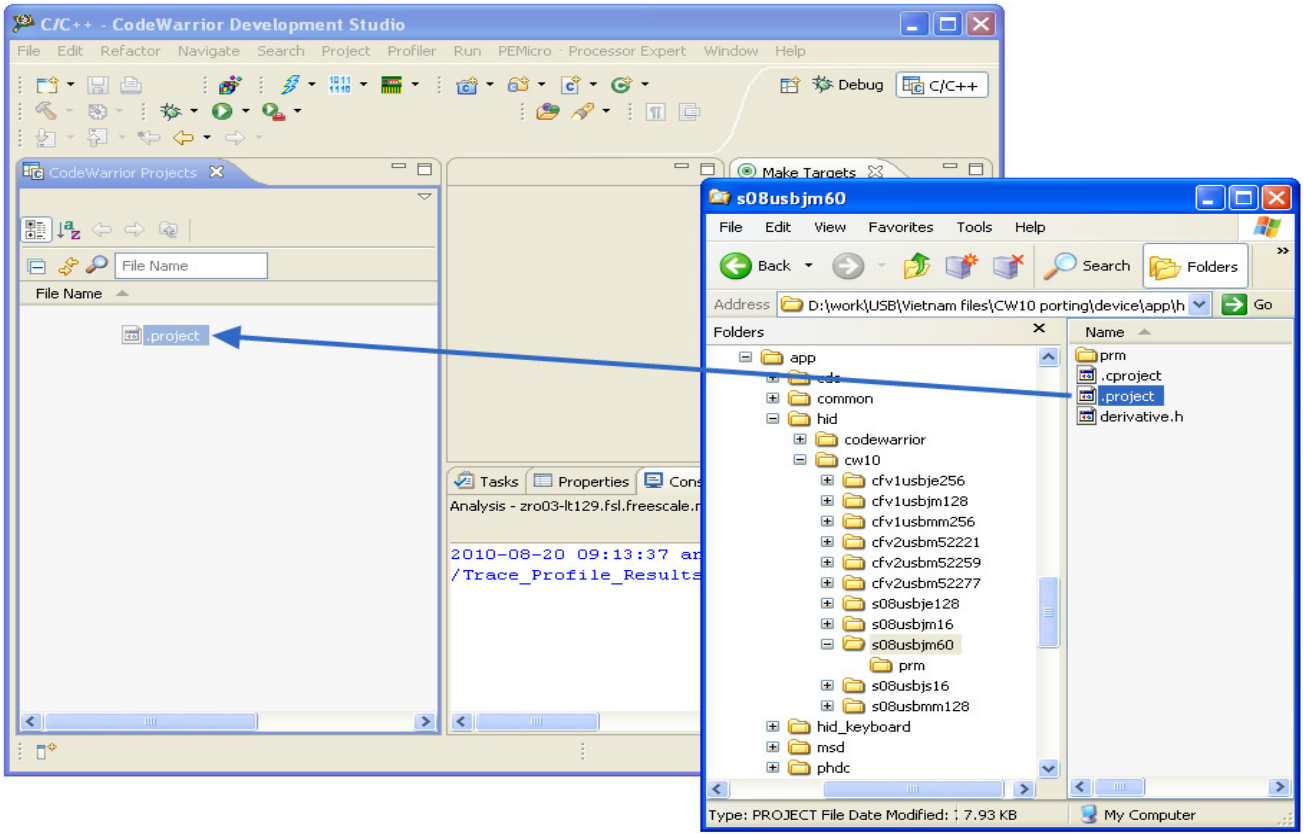
Before starting the process of building the project, make sure CodeWarrior 10 MCU is installed on your computer.

To build the (for example Kinetis k40/S08/CFV1/CFV2) project:

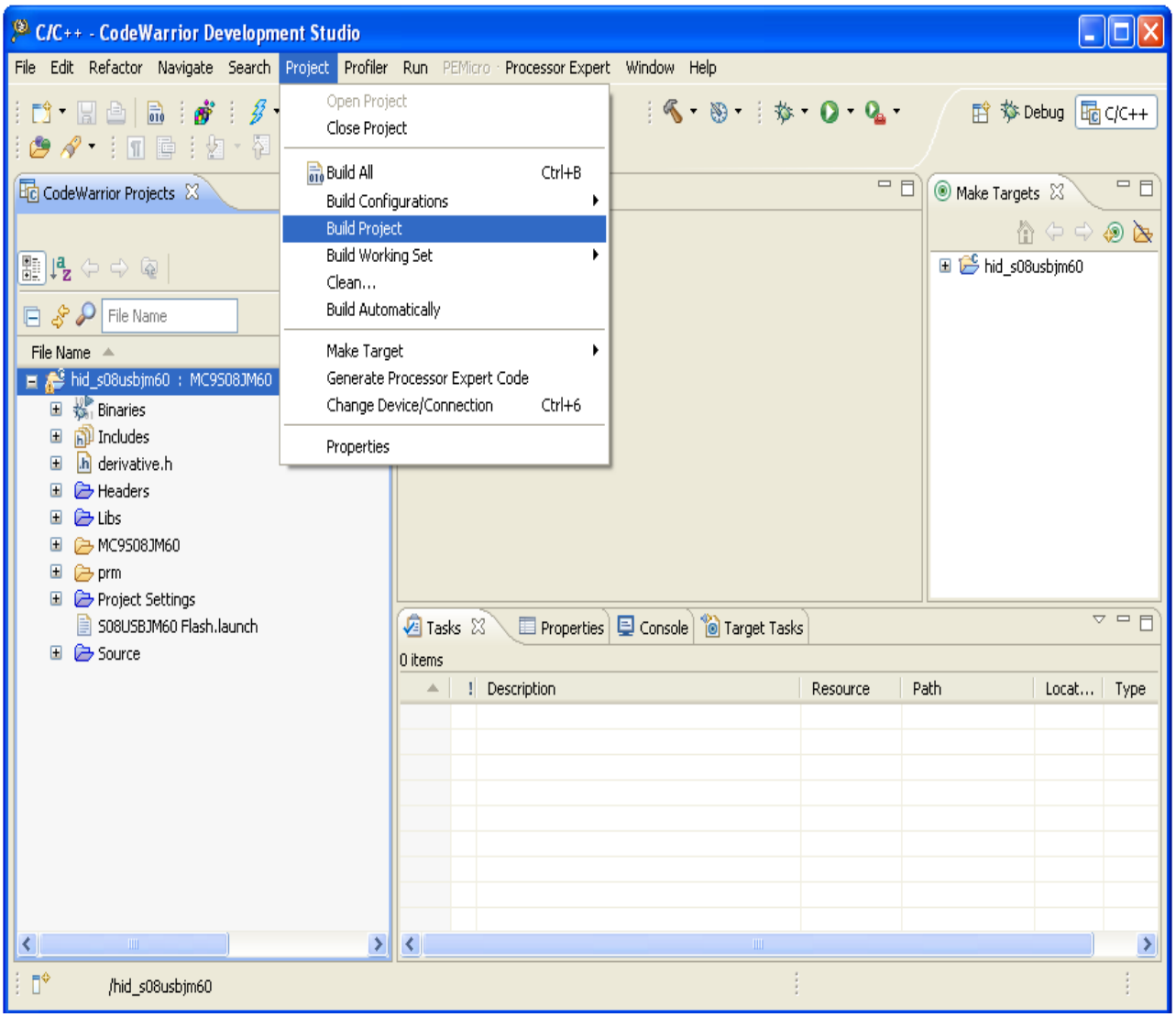
1. Navigate to the project folder (s08usbjm60) and locate the CodeWwarrior10 project file (.project).



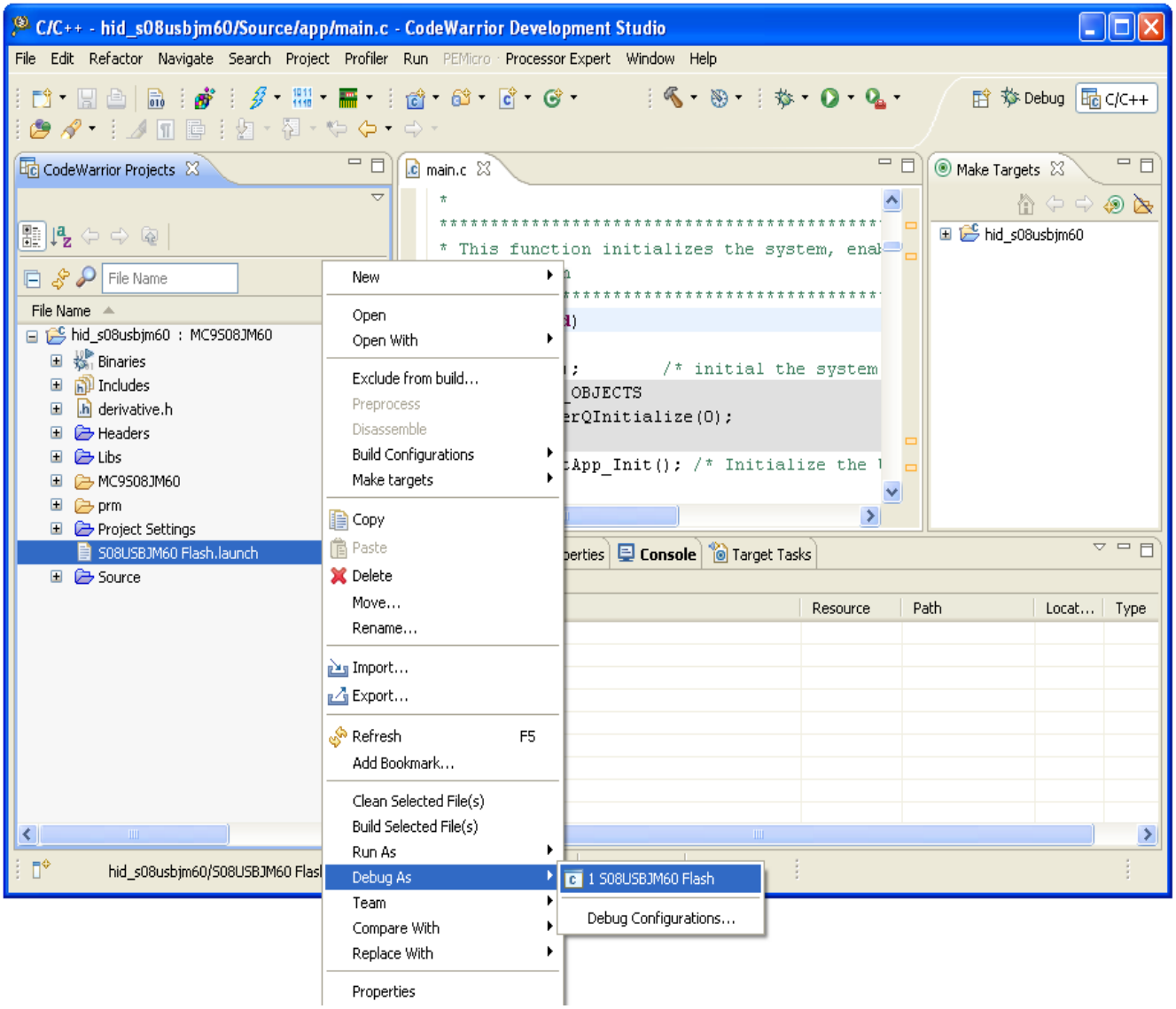
2. Open the project by dragging the .project file and dropping it into the CodeWarrior 10 project space.



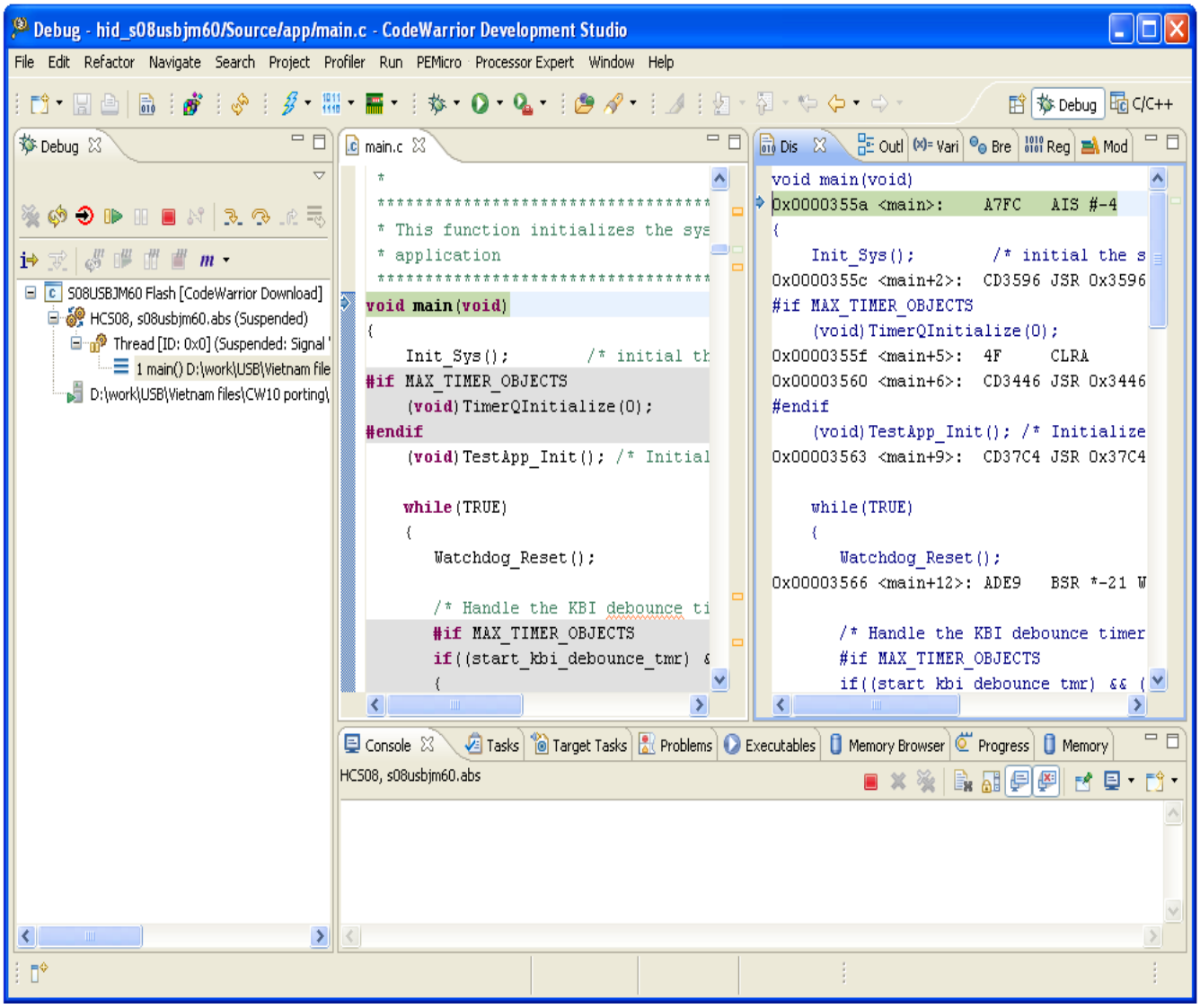
3. After you have opened the project, the following window appears. To build the project choose "Build Project" from the Project menu.



4. To run the application, first locate the S08USBJM60 Flash.launch conficuration in the current project space. Right-click it and choose Debug As > 1 S08USBJM60 Flash as in the window below.



5. After the image is programmed in the flash, the debugger window as shown in the next figure appears. Click on the Green arrow in the Debug tab to run the image.



A.2 Uninstall Freescale USB Stack Software

1. From your computer, click **Start** > **Settings** > **Control Panel** > **Add or Remove Programs**.

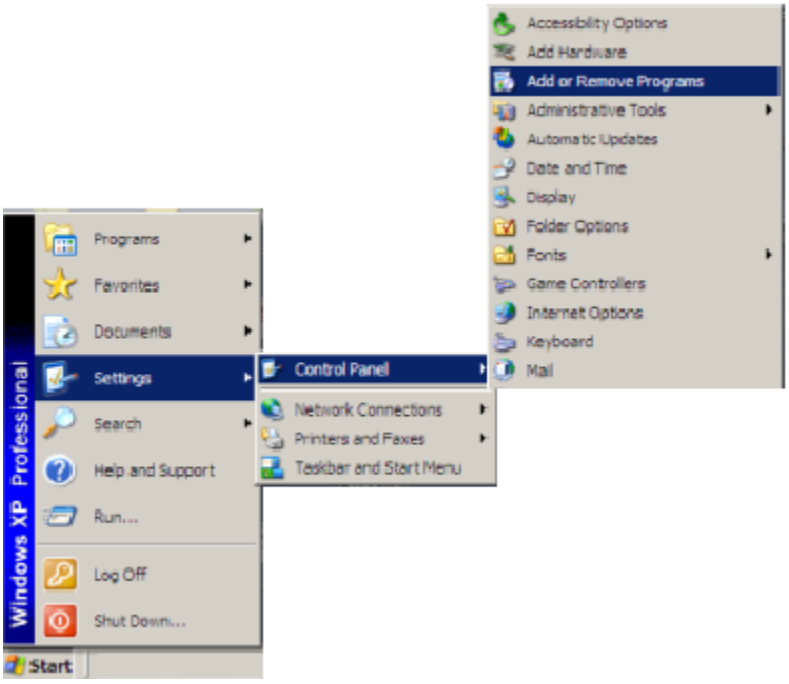


Figure 5-2. Add or Remove Programs launch from Control Panel

2. The following example shows the demonstration for uninstalling Freescale USB Stack. You can follow the same instructions for new versions.

Example:

1. In the Windows Control Panel "Add/Remove Programs Tool, select Freescale USB Stack and click on the Change/Remove button.
2. The uninstall confirmation message appears. Click on the **Yes** button to uninstall.

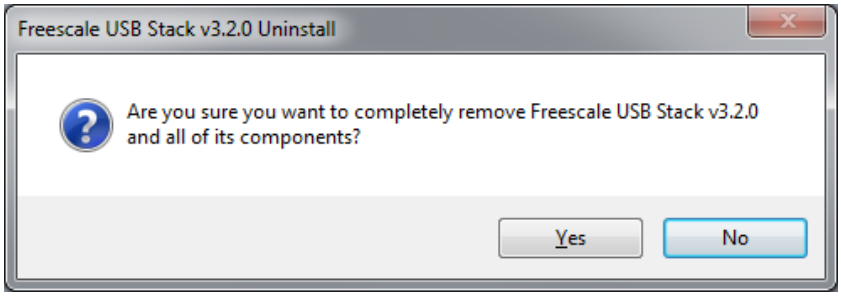


Figure A-15. Freescale USB Stack Uninstall confirmation message

3. A message box appears. Click on the **Ok** button to complete the uninstall operation.

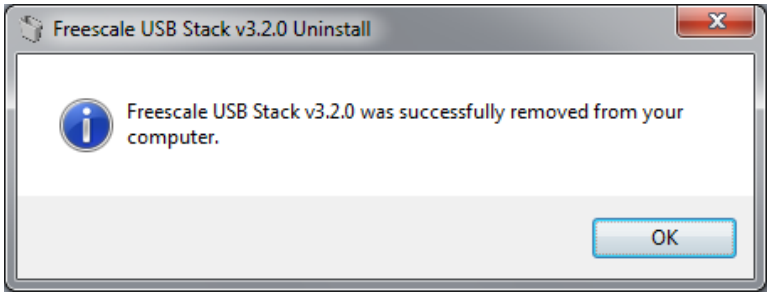


Figure A-16. Freescale USB Stack Uninstall completion message

A.3 Important files

Table A-1 shows the programming files that contain source code for classes.

Table A-1. Important files

Files	Description
device\source\class\usb_cdc.c	This is communication device functionality specific device class source code file.
device\source\class\usb_cdc.h	This is communication device functionality specific device class header file.
device\source\class\usb_cdc_pstn.c	This is communication PSTN sub-class source code file
device\source\class\usb_cdc_pstn.h	This is communication PSTN sub-class header file
device\source\class\usb_hid.c	This is human interface device functionality specific device class source code file.
device\source\class\usb_hid.h	This is human interface device functionality specific device class header file.
device\source\class\usb_phdc.c	This is personal healthcare device functionality specific device class source code file.
device\source\class\usb_phdc.h	This is personal healthcare device functionality specific device class header file.
device\source\common\usb_class.c	This is class independent source code file.
device\source\common\usb_class.h	This is class independent header file.
device\source\common\usb_framework.c	This is USB specification chapter 9 request handing source code file.
device\source\common\usb_framework.h	This is USB specification chapter 9 request handing header file.
device\source\driver\cfv1\usb_bdt.h	This is controller specific header file containing buffer descriptor table structure for CFV1 devices.
device\source\driver\cfv1\usb_dci.c	This is controller specific low-level driver source code file for CFV1 devices.
device\source\driver\cfv1\usb_dci.h	This is controller specific low level driver header file for CFV1 devices.
device\source\driver\cfv1\usb_dciapi.h	This file contains DCI API function definitions for CFV1 devices.
device\source\driver\s08\usb_bdt.h	This is controller specific header file containing buffer descriptor table structure for S08 devices.
device\source\driver\s08\usb_dci.c	This is controller specific low-level driver source code file for S08 devices.
device\source\driver\s08\usb_dci.h	This is controller specific low level driver header file for S08 devices.
device\source\driver\s08\usb_dciapi.h	The file contains DCI API function definitions for S08 devices.
device\source\driver\usb_devapi.h	This is the header file defining low-level driver interfaces.
device\source\driver\usb_driver.c	This is the USB stack driver interface source code file.
device\source\class\ usb_msc.c	This is mass storage device functionality specific device class source code file.
device\source\class\ usb_msc.h	This is mass storage device functionality specific device class header file.
device\source\class\ usb_msc_scsi.c	This is mass storage SCSI sub-class source code file.
device\source\class\ usb_msc_scsi.h	This is mass storage SCSI sub-class header file.

Appendix B Human Interface Device (HID) Demo

B.1 Setting up the demo

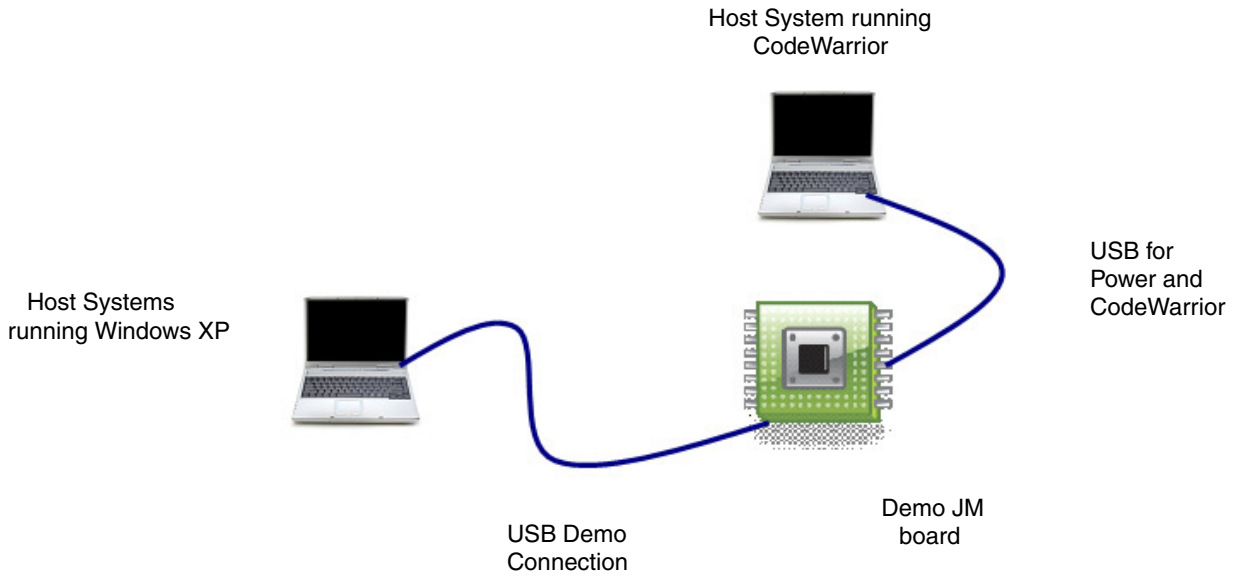


Figure B-1. HID demo setup

Figure B-1 describes the demo setup. DemoJM is connected to two personal computers using USB cables. The first computer is used to supply power to the board and is used to program the image to the flash. The second computer is used as the host system where the USB host driver resides. Although, Figure B-1 shows two computers, the connection can also be achieved using one computer with two USB ports.

B.2 Running the demo

After the HID application is programmed into the silicon flash, the demo can be run using the following procedure.

1. Connect the hardware to the Windows host computer. As soon as you turn on the device, the HID device gets installed onto the Windows host computer. You must see the callout as shown in Figure B-2 on the right bottom corner of your screen. At this point, the windows installs the host driver for the device.

Figure B-2. Find New Hardware callout

2. To verify whether the mouse has been properly installed or not, you must see the JM device entry in the device manager.

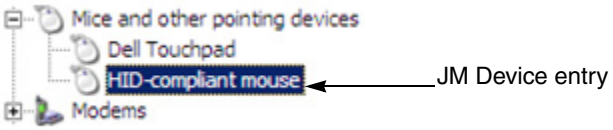


Figure B-3. JM device entry

3. After the HID device is installed, it can be moved on the host computer screen by pressing the push buttons. Figure B-4 shows the function of these buttons.

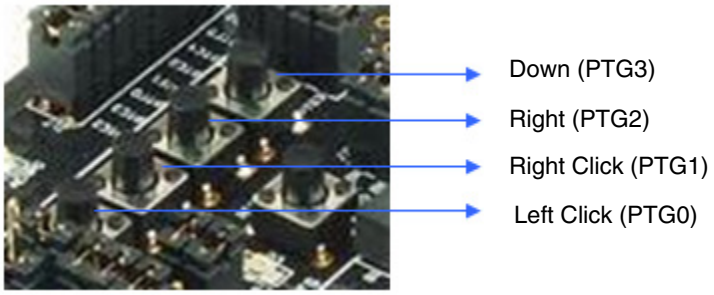


Figure B-4. HID device push buttons

NOTE

For JS16, Left Click (PTG0) push button is not avaialble for use.

Appendix C Personal Healthcare — Multi-Specialization Device Demo

Personal healthcare application interacts with the host system using IEEE-11073 – 20601 and (IEEE-11073 – 10415 (weigh scale), IEEE-11073 – 10407 (Blood Pressure Monitor), IEEE-11073 – 10417 (Glucose Meter), and IEEE-11073 – 10408 (Thermometer) protocol. To run this demo, a host system is required that runs the same IEEE-11073 protocols. One example of such implementation is done by Continua Alliance. In this demo, Continua Manager is used on the host system.

C.1 Setting up the demo

Set the systems as described in the previous section (HID demo).

1. Get the Continua Alliance (www.continuaalliance.org) CESL Reference Software V1.0 RC1.
2. Install the software on a host system
3. Program the JM60 flash with the PHDC multi-specialization application using CodeWarrior IDE.

NOTE

CESL reference software is not provided as part of the suite. You will have to get this software independently from Continua Alliance.

C.2 Running the demo

After the system has been set, you must follow these steps to run the demo:

1. Turn on the DemoJM board. Found New Hardware window appears.

Figure C-1. Found New Hardware window

2. Select **Install from a list or specific location (Advanced)** option as shown in Figure C-1, and click on the **Next** button. Search and installation options window appears as shown in Figure C-2.

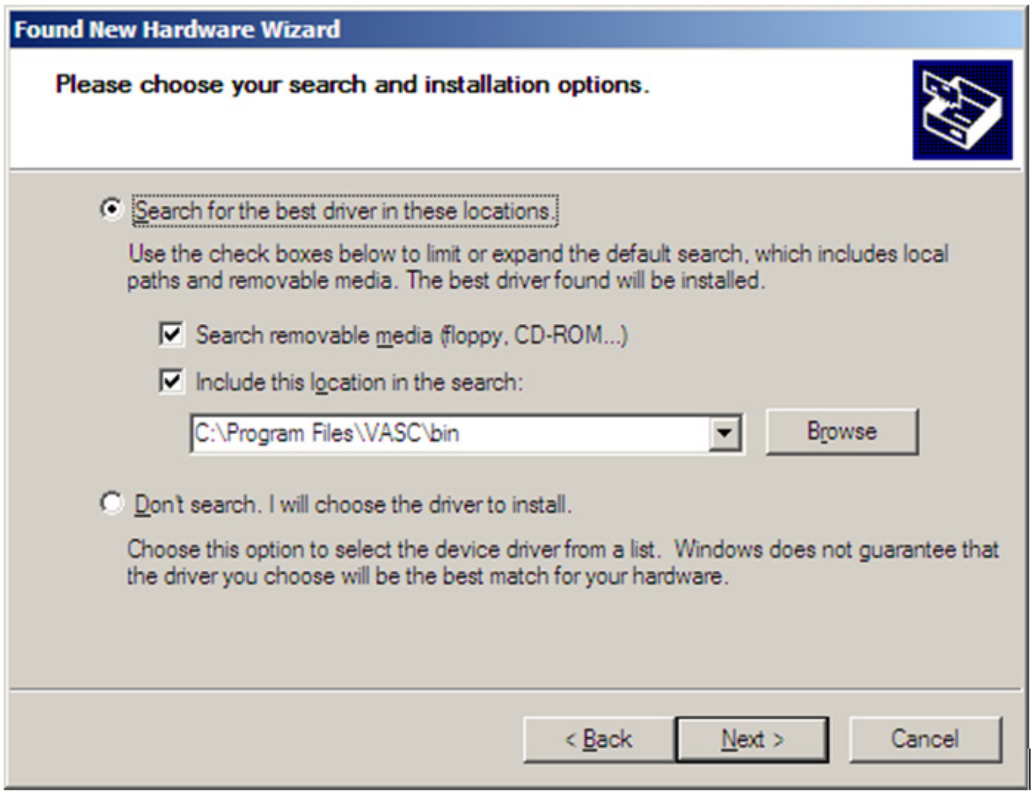


Figure C-2. Search and installation options

Point the search path to the bin directory where the Continua CESL software was installed and click on the **Next** button. The driver for the device will get installed.

To verify the installation, open the device manager. You must see the Continua USB Interface device entries.

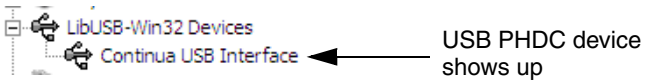


Figure C-3. Continua USB PHDC Device Entry in Device Manager

3. Launch the Continua Manager from **Start** > **All Programs** menu as shown in Figure C-4.

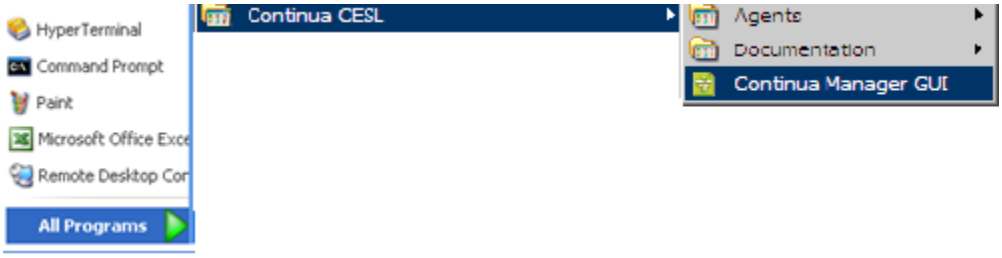


Figure C-4. Launch Continua Manager

4. The Continua Manager GUI opens as shown in Figure C-5. Enter the name of the skim directory and click on the **Start Transport** button.

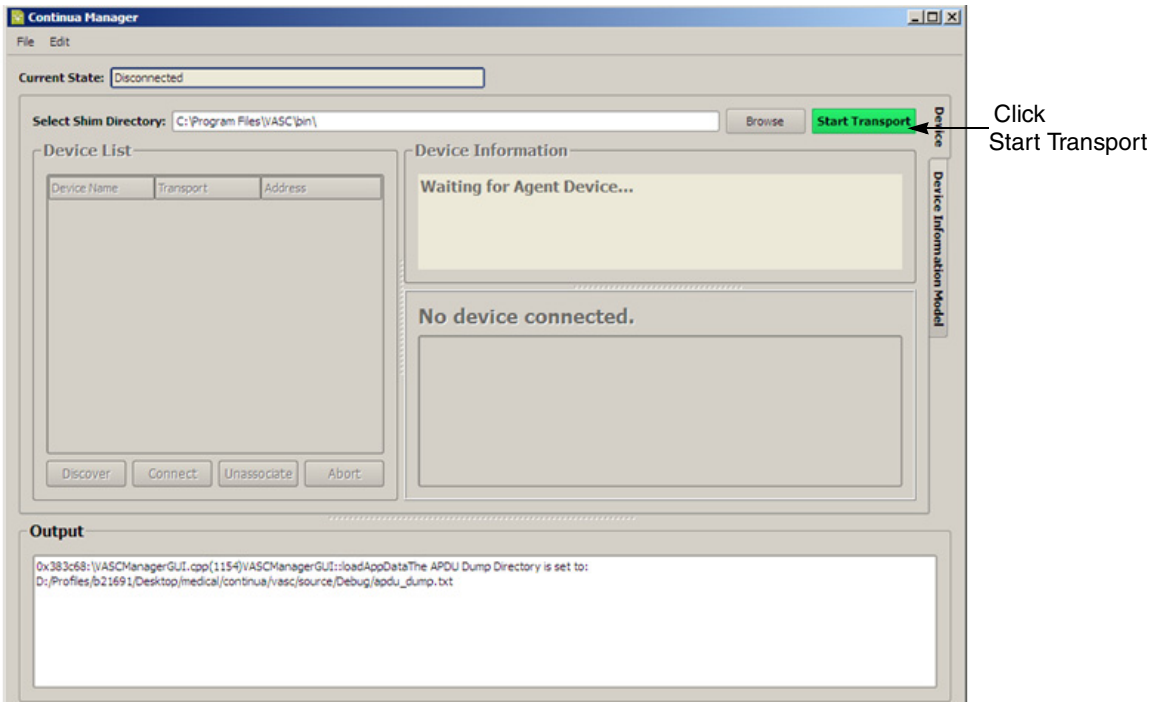


Figure C-5. Continua Manager window

5. S 3 Medic Event connection confirmation pop-up appears as shown in the Figure C-6. Click on the **Yes** button to continue.

Figure C-6. Connect to S3 Medic Agent

6. The Continua Manager now enters the Operating State using default specialization (weigh scale). The Continua application window appears as shown in Figure C-7.

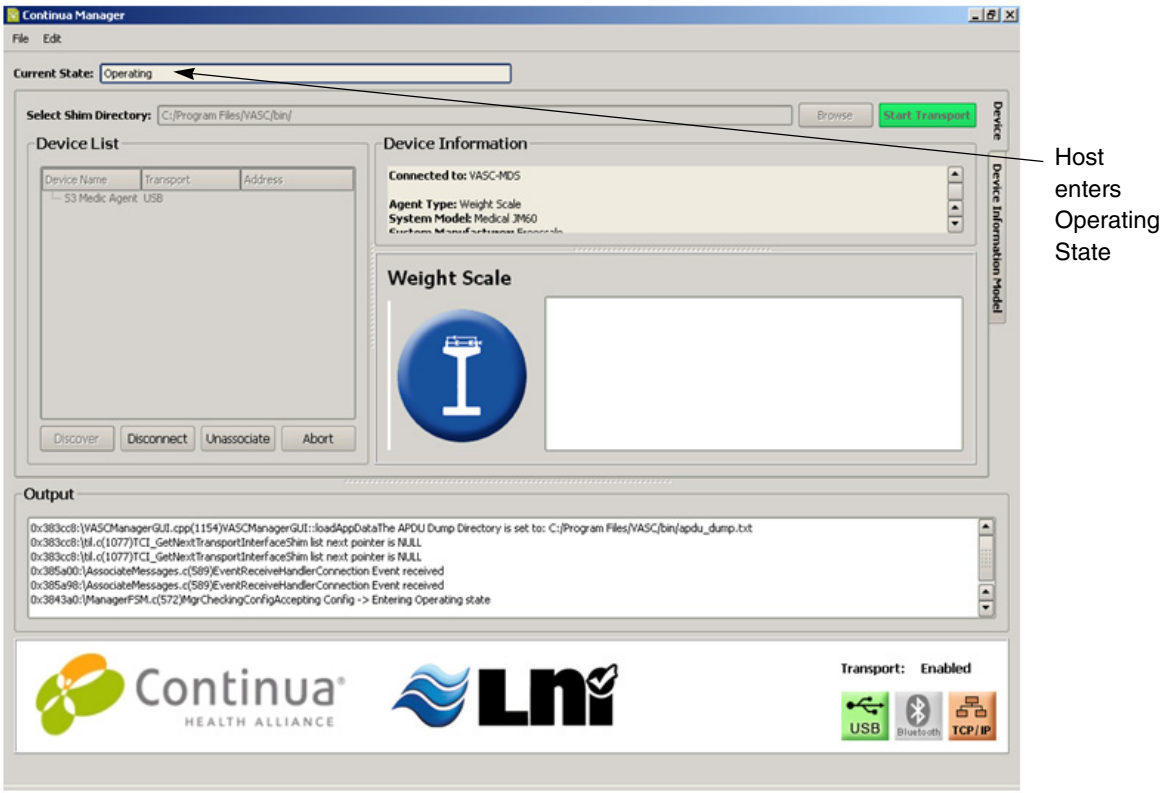


Figure C-7. Host entering operating state

7. After the host device is in operating state, Push Buttons on the device can be used to send weight measurements to the host. Figure C-8 shows the function of these buttons.

Figure C-8. DemoJM push button panel

8. When the push button to send the measurement is pressed, measurements are sent to the Continua Host Manager as shown in Figure C-9.

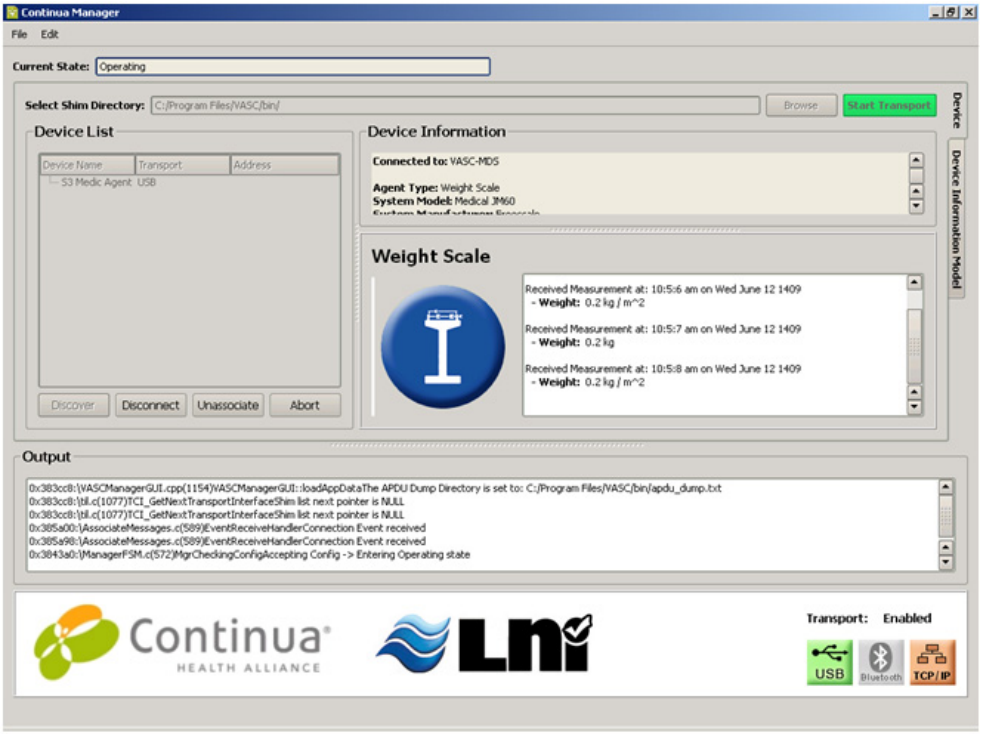


Figure C-9. Weigh scale device detection with Continua Host

9. When **Select Config (PTG3)** push button is pressed, device specialization changes. The device reconnects with the Continua host with a new specialization. Selected specialization is displayed on LED panel on DemoJM board. The device initiates connection sequence with Continua Host using new specialization after 3 seconds.

   The device specializations are preprogrammed in the device in the following order:

Selection ID	Configuration
0	Weigh Scale (Default)
1	Glucose Meter
2	Blood Pressure Monitor
3	Thermometer

Figure C-10 shows the DemoJM LED panel.

Figure C-10. DemoJM LED display panel

For JS16 board, only PTE2 and PTE3 are available for use. PTE2 and PTE3 LED display for different specializations are:

Configuration	PTE2	PTE3
Weigh Scale	OFF	OFF
Glucose Meter	OFF	ON
Blood Pressure Meter	ON	OFF
Thermometer	ON	ON

For JM16, JM60, and MCF51JM128, all LEDs on DemoJM board are available for use. PTD2, PTF5, PTC4, and PTC2 LED display for different specializations are:

Configuration	PTD2	PTF5	PTC4	PTC2
Weigh Scale	ON	OFF	OFF	OFF
Glucose Meter	OFF	ON	OFF	OFF
Blood Pressure Meter	OFF	OFF	ON	OFF
Thermometer	OFF	OFF	OFF	ON

10. When a particular Configuration is selected, and the Continua host comes to an Operating State, measurements can be sent using **Send Measurement (PTG1)** push button.

Figure C-11, Figure C-12, Figure C-13, and Figure C-14 show various device specializations detected on Continua host:

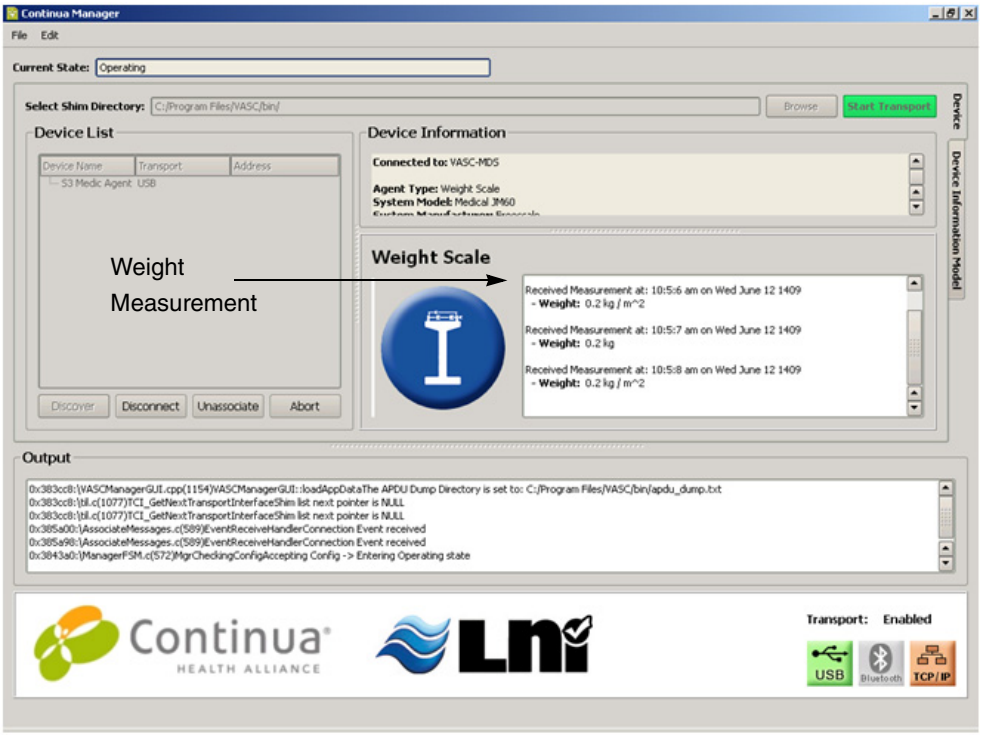


Figure C-11. Weigh scale device detection with Continua Host

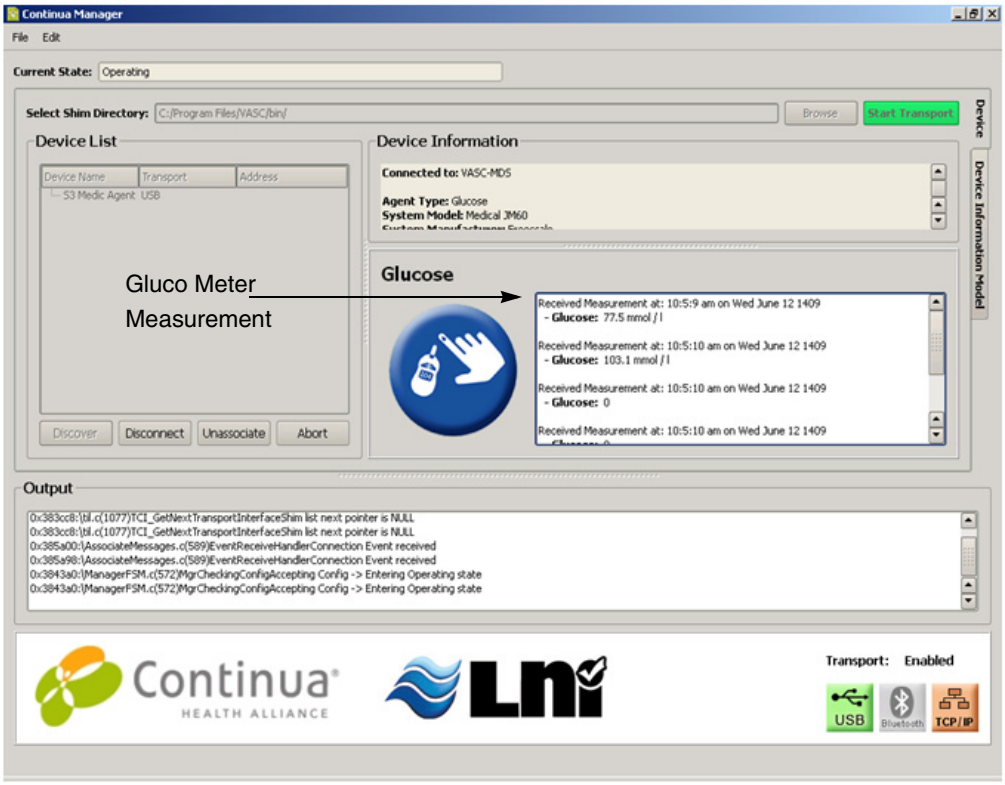


Figure C-12. Glucose meter device detection with Continua Host

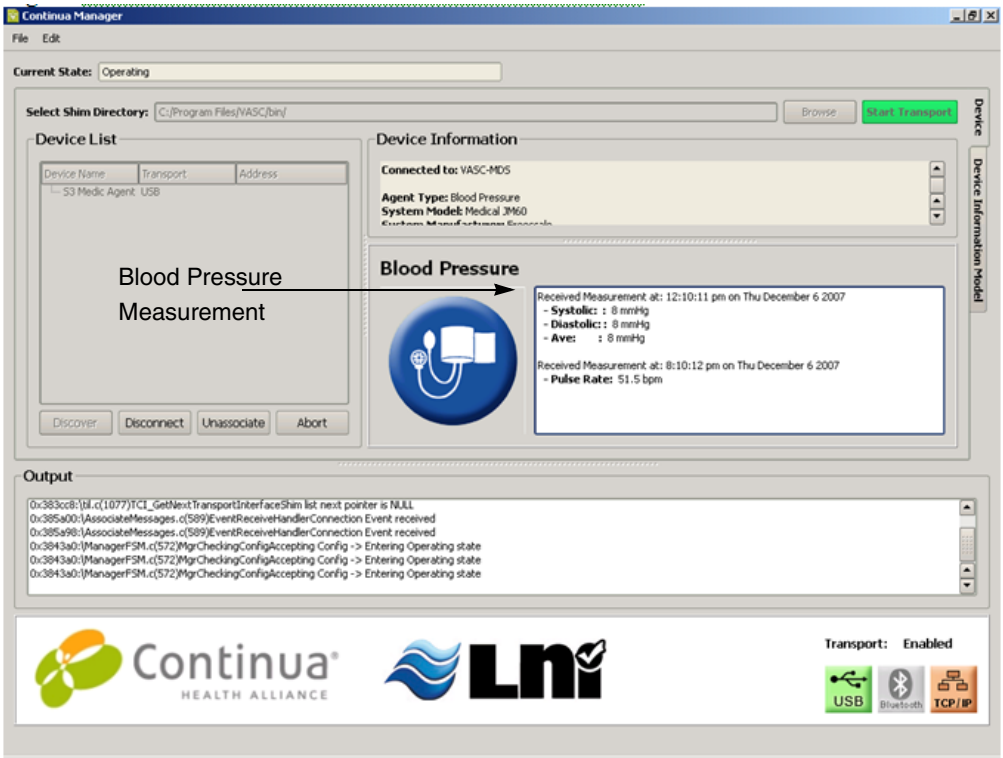


Figure C-13. Blood Pressure monitor device detection with Continua Host

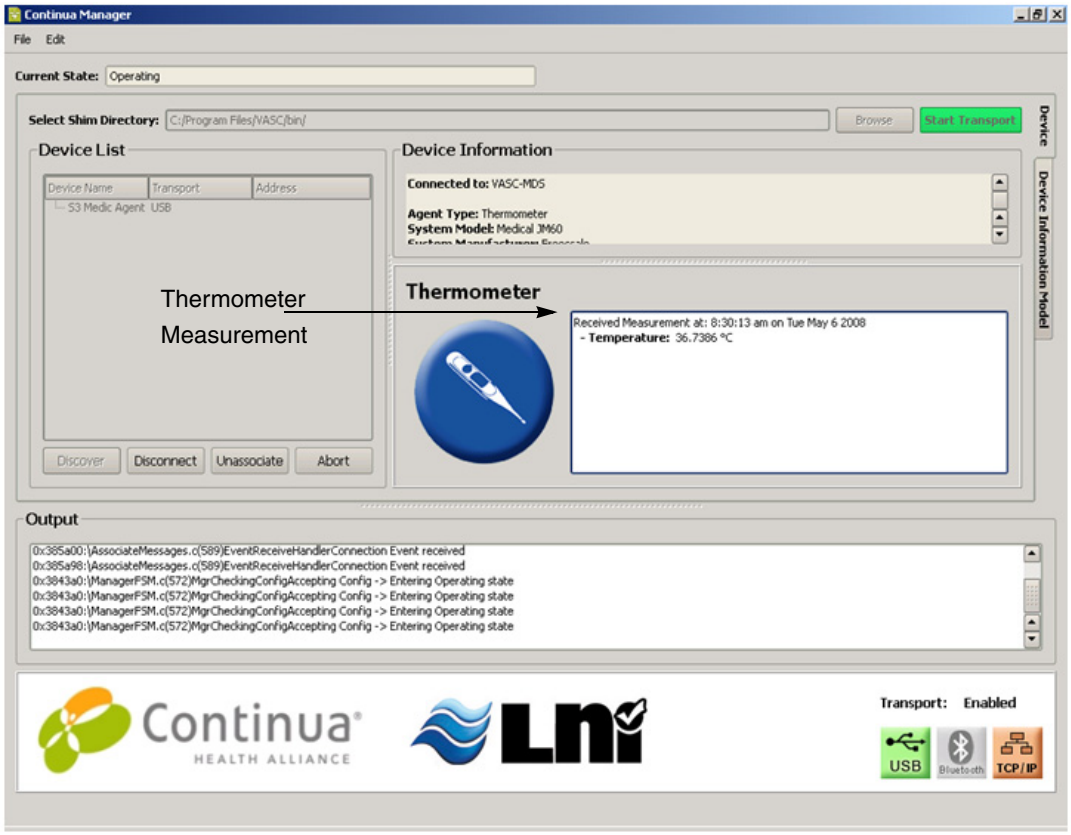


Figure C-14. Thermometer device detection with Continua Host

Appendix D Human Interface Device (HID) Demo

D.1 Setting up the demo

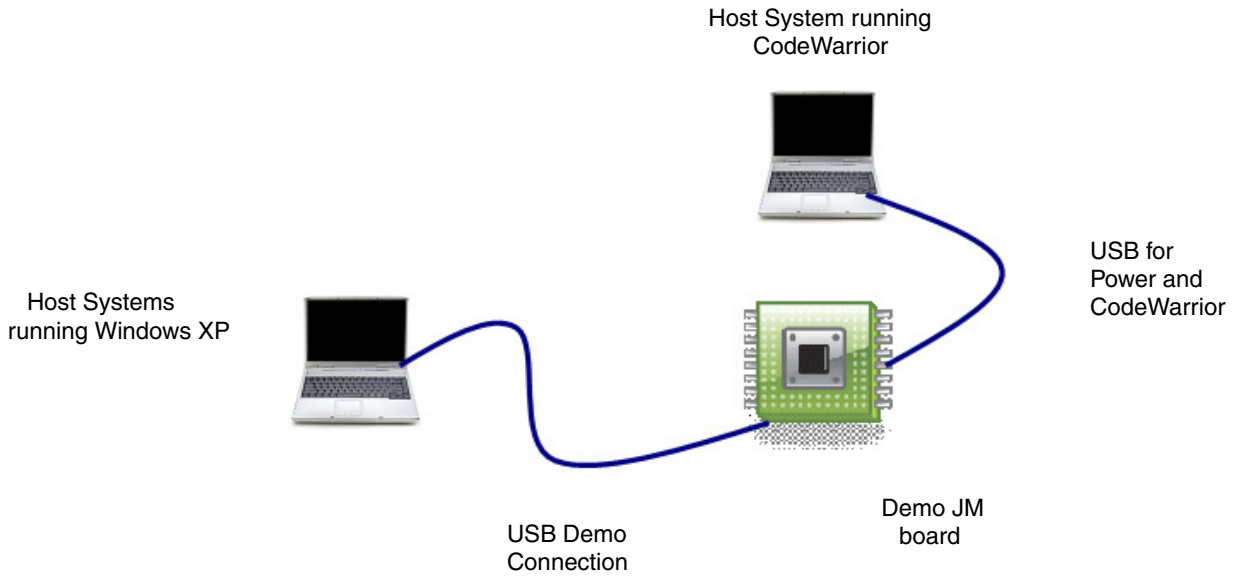


Figure D-1. HID demo setup

Figure D-1 describes the demo setup. DemoJM is connected to two personal computers using USB cables. The first computer is used to supply power to the board and is used to program the image to the flash. The second computer is used as the host system where the USB host driver resides. Although, Figure D-1 shows two computers, the connection can also be achieved using one computer with two USB ports.

D.2 Running the demo

After the HID application is programmed into the silicon flash, the demo can be run using the following procedure.

1. Connect the hardware to the Windows host computer. As soon as you turn on the device, the HID device gets installed onto the Windows host computer. You must see the callout as shown in Figure D-2 on the right bottom corner of your screen. At this point, the windows installs the host driver for the device.

Figure D-2. Find New Hardware callout

2. To verify whether the mouse has been properly installed or not, you must see the JM device entry in the device manager.

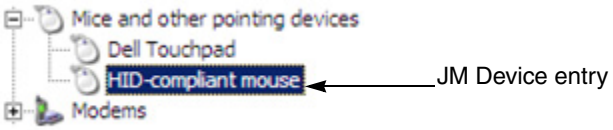


Figure D-3. JM device entry

3. After the HID device is installed, it can be moved on the host computer screen by pressing the push buttons. Figure D-4 shows the function of these buttons.

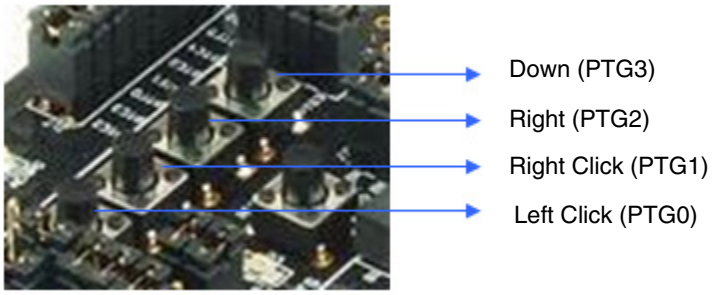


Figure D-4. HID device push buttons

NOTE

For JS16, Left Click (PTG0) push button is not avaialble for use.

Appendix E Personal Healthcare – Weigh Scale Device Demo

Personal healthcare application interacts with the host computer using IEEE-11073 – 20601 and IEEE-11073 – 10415 (weigh scale) protocols. To run the demo, the host computer runs the same IEEE-11073 protocols. One example of such implementation is covered by Continua Alliance. In our demo, we have used Continua Manager on the host computer.

E.1 Setting up the demo

Set the systems as described in the Appendix D, "Human Interface Device (HID) Demo."

1. Install Continua Alliance (www.continuaalliance.org) enabled PC software such as the Lamprey Networks Inc. CESL or HealthLink
2. Install the software on the host computer.
3. Program the microcontroller flash with the PHDC application using CodeWarrior IDE.

NOTE

Continua Alliance enabled PC software is not provided as part of the suite. You will have to get this software independently from LNI.

E.2 Running the demo

After the system has been set, you must follow these steps to run the demo:

1. Turn on the DemoJM board. Found New Hardware window appears.

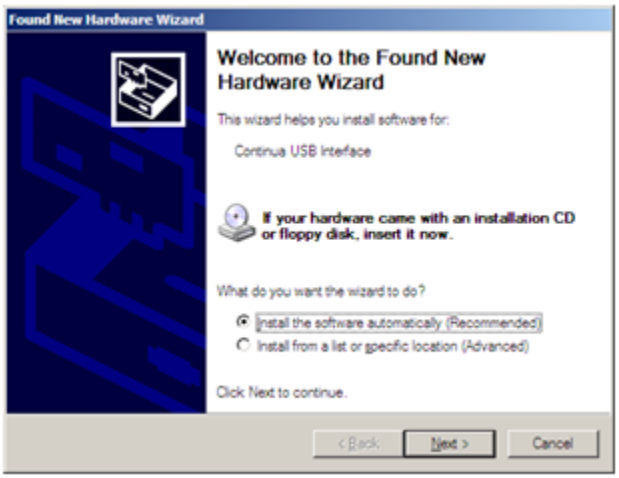


Figure E-1. Found New Hardware window

2. Select **Install from a list or specific location (Advanced)** option as shown in Figure E-1, and click on the **Next** button. Search and installation options window appears as shown in Figure E-2.

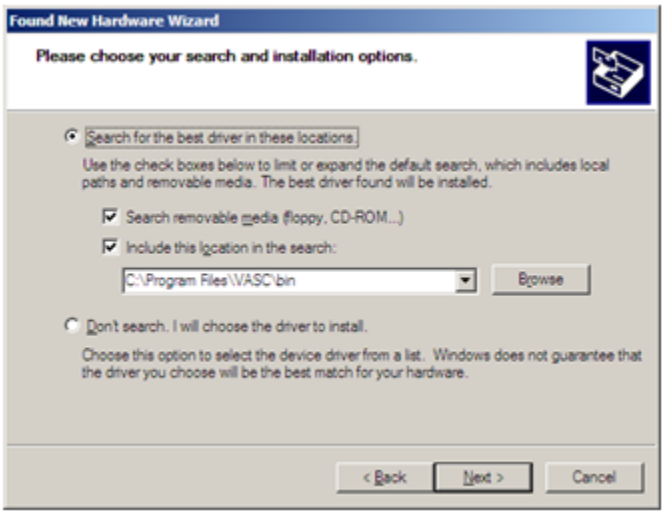


Figure E-2. Search and Installation options

Point the search path to the bin directory where the Continua CESL software was installed and click on the **Next** button. The driver for the device will get installed.

To verify the installation, open the device manager. You must see the USB PHDC device entries.

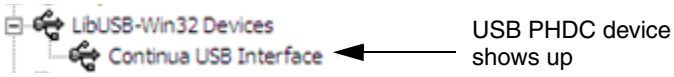


Figure E-3. USB PHDC Device Entry in Device Manager

3. Launch the Continua Manager from the **Start** > All **Programs** menu towards the left-bottom side of your computer.

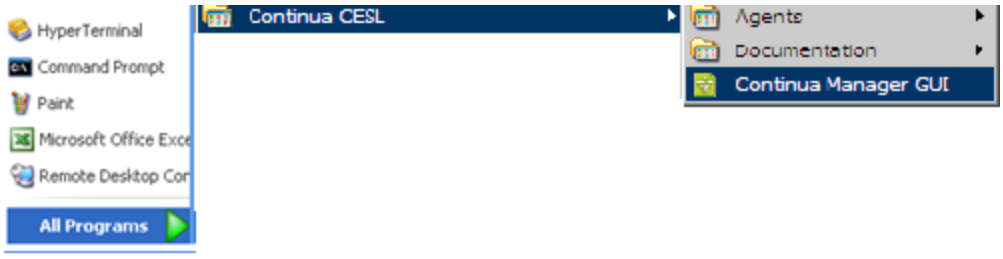


Figure E-4. Launch Continua Manager

You must see the following application window.

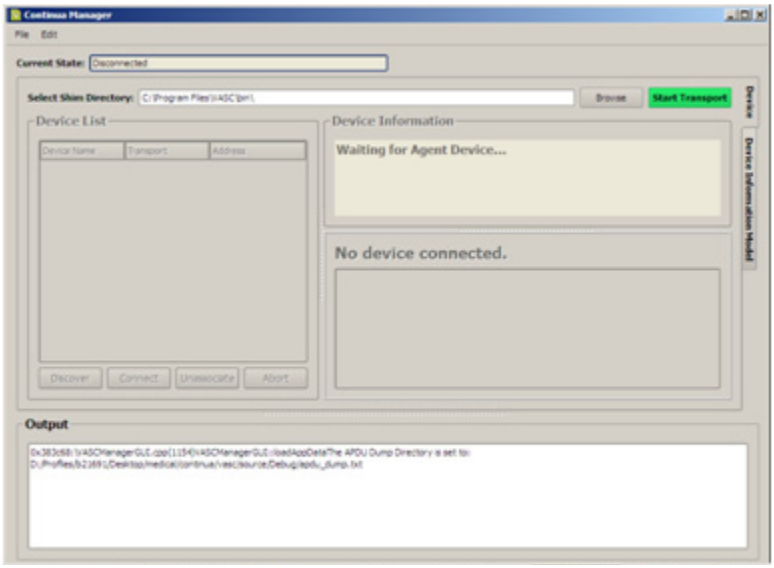


Figure E-5. Continua Manager window

4. Click on the **Start Transport** button. S 3 Madic Event connection confirmation pop-up appears as shown in Figure E-6. Click on the **Yes** button to continue.

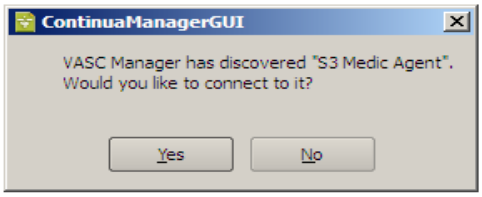


Figure E-6. Continua Manager GUI

5. The Continua Manager now enters the operating state. The Continua application window appears as shown in Figure E-7.

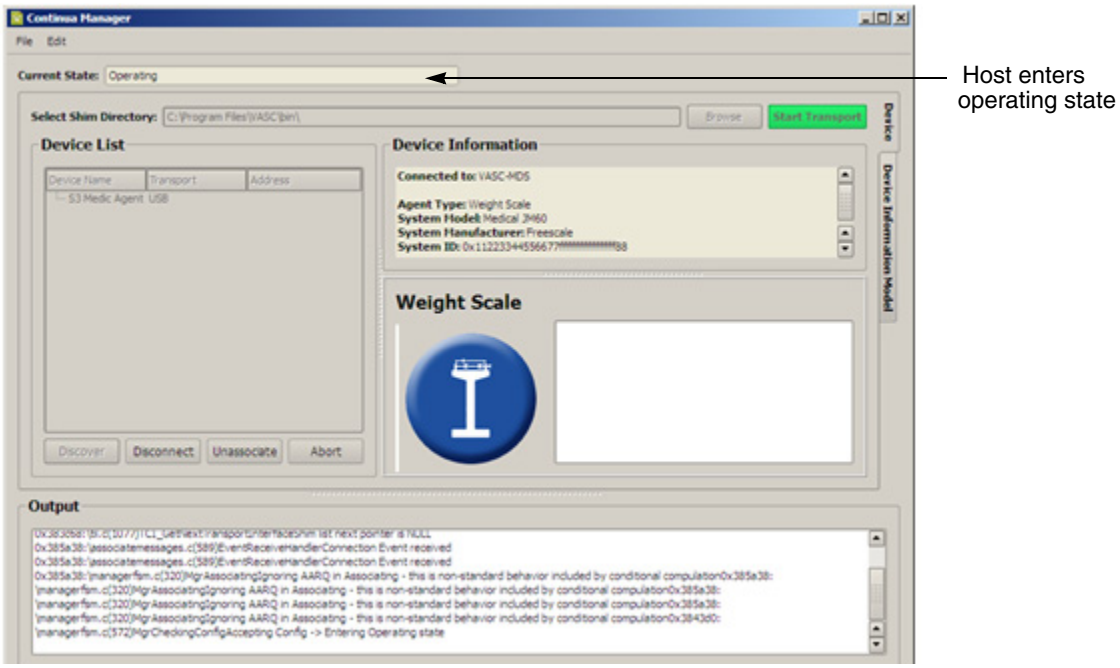


Figure E-7. Continua Application

6. After the host device is in operating state, the push buttons on the device can be used to send weight measurements to the host. Figure E-8 shows the function of these buttons.

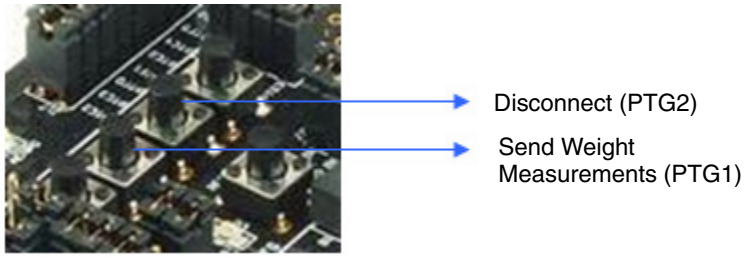


Figure E-8. PHCD push buttons

7. When the push button to send the measurement is pressed, the measurements are sent to the Continua Host Manager.

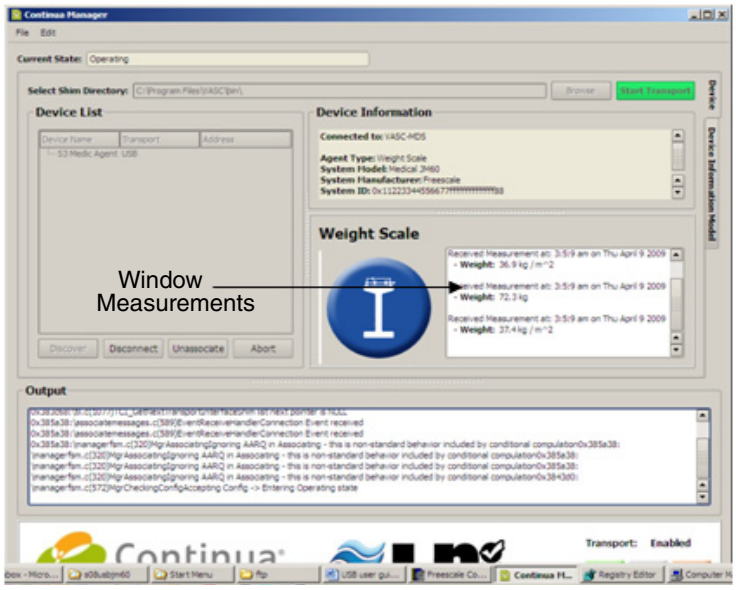


Figure E-9. Continua Manager

Appendix F SD Card Demo

The SD Card demo implements the SCSI subclass of the USB MSC class. SD Card is attached to DEMOFLEXISJMSD board (board maps SD interface to SPI and is specially designed for this purpose). On running the application, SD Card is available as removable disk in Windows.

To select SD Card Application, set SD_CARD_APP macro in user_config.h to 1 and RAM_DISK_APP to 0.

F.1 Setting up the demo

Set the systems as described in the Appendix D, "Human Interface Device (HID) Demo."

F.2 Running the demo

After the SD Card application is programmed into the silicon flash, you must follow these steps to run the demo:

1. Connect the hardware to Microsoft Windows host computer. As soon as you turn on the device, the MSD device gets installed onto the host computer. You must see the callout as shown in Figure F-1 on the right bottom corner of your screen. At this point, Microsoft Windows installs the host driver for the device.

Figure F-1. Found New Hardware callout

2. To verify whether the SD Card has been properly installed (or detected by Microsoft Windows), you must see the JM device entry in the device manager.

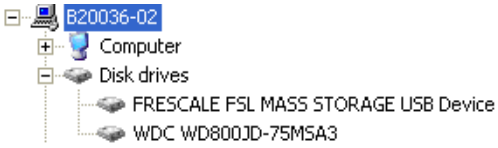


Figure F-2. JM device entry

3. After the MSD device is installed, you can perform read, write, and format operations on SD Card.

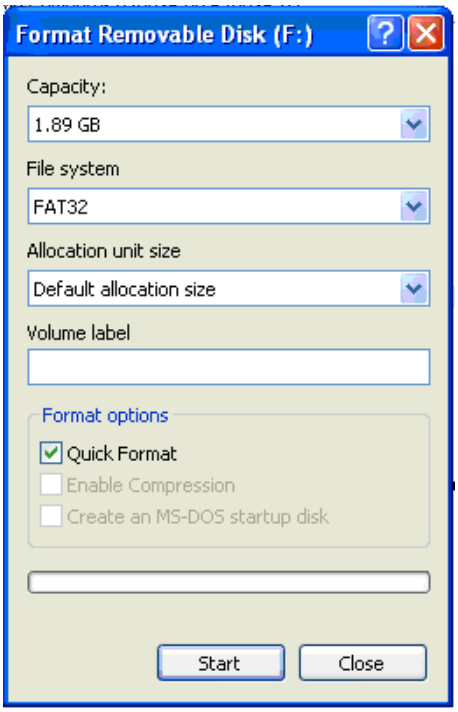


Figure F-3. Option to format the SD card

NOTE

Because of SPI interface, all operations on SD Card are very slow for this demo application.

Appendix G USB Audio Demo

This section explains how to use the USB Audio demo software package. The USB Audio demo is developed based on USB audio class consisting of two different applications:

- **USB audio speaker**—Receives audio stream data from host and plays it
- **USB audio generator**—Sends audio data stream to the host

Both of these applications also support specific requests from host such as Mute Control, Volume Control, and many more. The demo board used is DemoJM board (DemoJM128).

G.1 Audio speaker demo

G.1.1 Setting up the demo

Figure G-1 describes the demo setup. The DemoJM board is connected to a PC (host) using two USB cables and one speaker through an external circuit. The PC uses one USB cable to supply power to the board and program the image to the flash. The PC also acts as the host system and the DemoJM board acts as the USB device. They are connected by the second USB cable. A COM connection is also established to display the log of the DemoJM application.

The external circuit (Figure G-2) is connected to the pin PTF0/TPM1CH2 of DemoJM board (it corresponds to pin 25 on DemoJM128) and is used to filter the audio signal before entering the speaker.

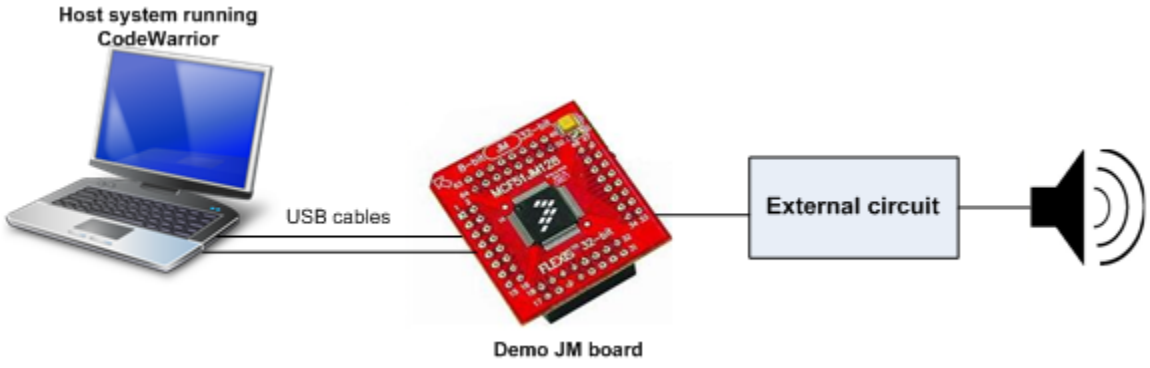


Figure G-1. Audio speaker demo setup

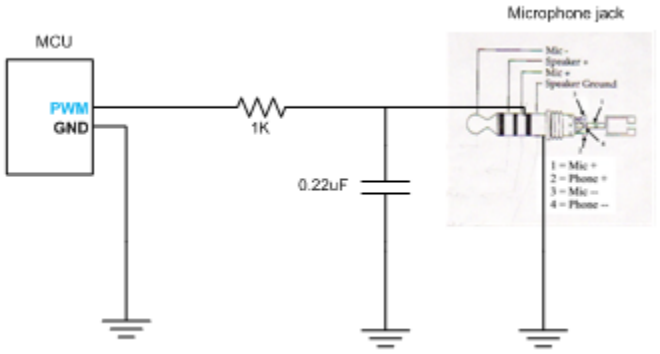


Figure G-2. External circuit

G.1.2 Running the demo

After the system has been set, you must follow these steps to run the demo:

1. Plug USB Audio Device in to the PC. As soon as you turn on the device, it is recognized by the host and is installed automatically. You must see the callout as shown in Figure G-3 on the right bottom corner of your screen.

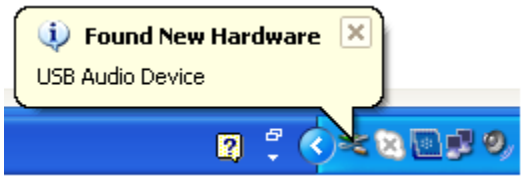


Figure G-3. Find New Hardware Callout

2. After successful installation, the host indicates that the device is ready to use as shown in Figure G-4

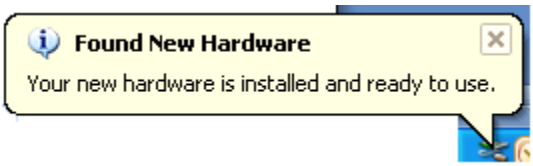


Figure G-4. Installation of USB Audio Device

3. To verify whether the USB Audio Device has been installed properly or not, you must see the device entry in the device manager as shown in Figure G-5.

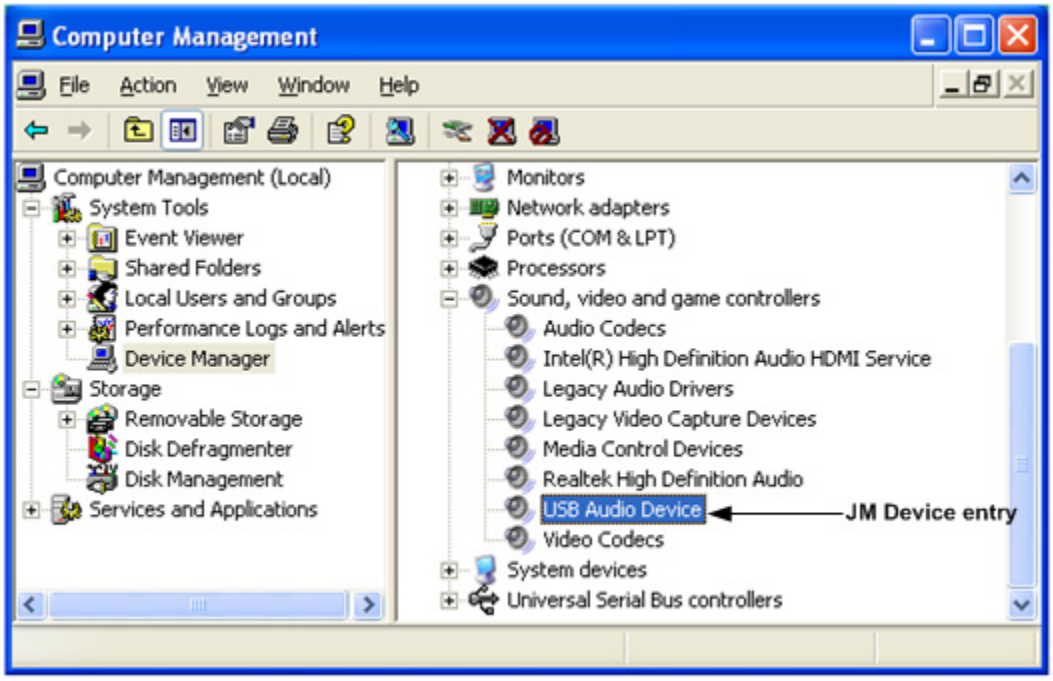


Figure G-5. Device manager dialog

4. Double-click on the USB Audio Device icon, USB Audio Device Properties dialog appears as shown in Figure G-6

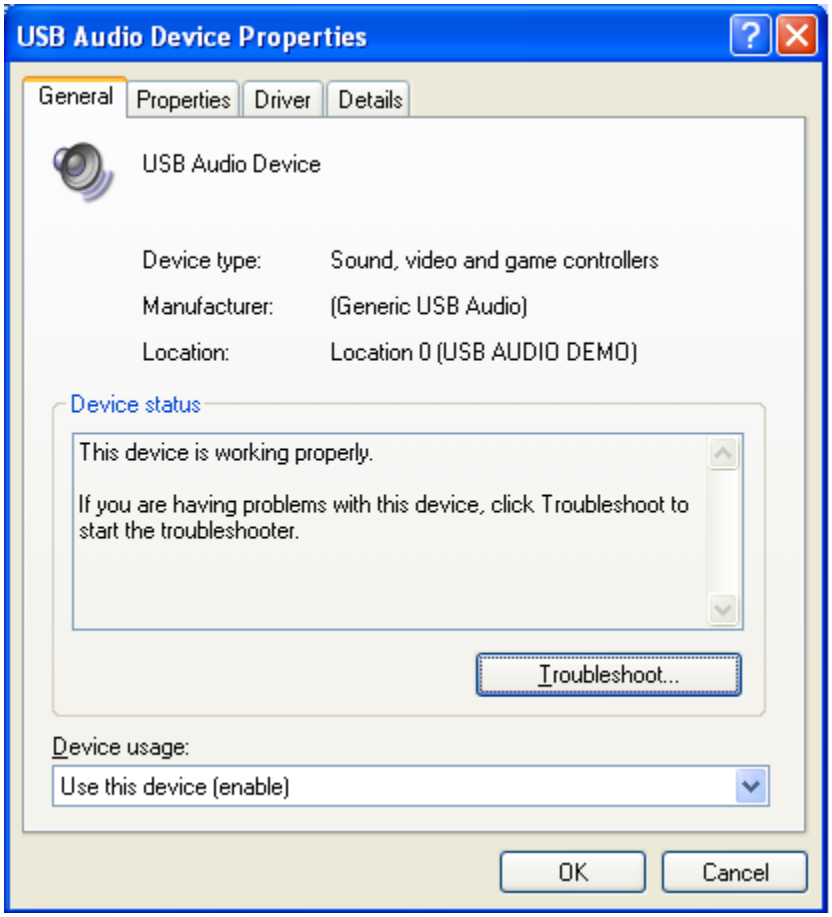


Figure G-6. USB Audio Device Properties dialog

5. To verify whether the USB Audio Device has been selected as the default device or not, you must right-click on master volume icon. Master volume dialog appears as shown in Figure G-7.

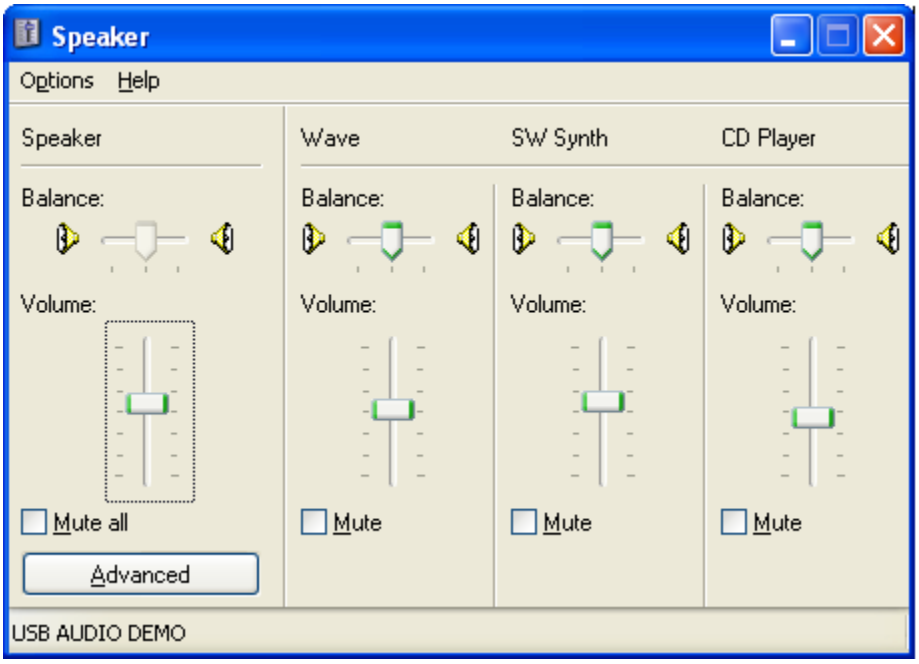


Figure G-7. Master volume

6. After the installation, the device works as a sound driver and PC can control it. To show that the value is already adjusted, the application sends the received data back to PC via UART. On PC side, the data is captured by using HyperTerminal software.
   a) Open the HyperTerminal application as shown in Figure G-8.

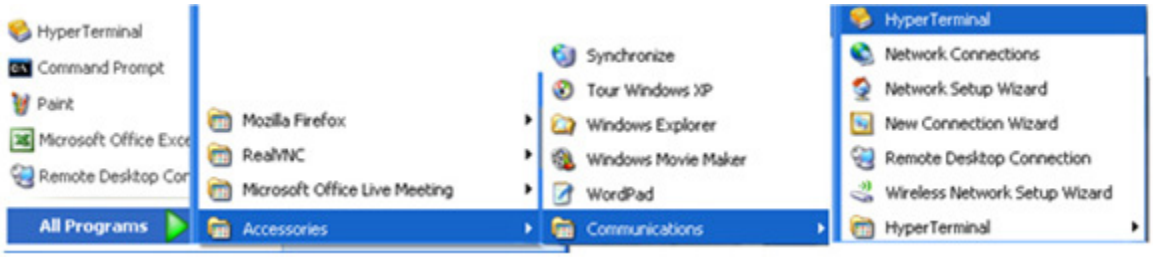


Figure G-8. Launch HyperTerminal application

b) The HyperTerminal application is shown in Figure G-9. Enter the name of the connection and click OK button.

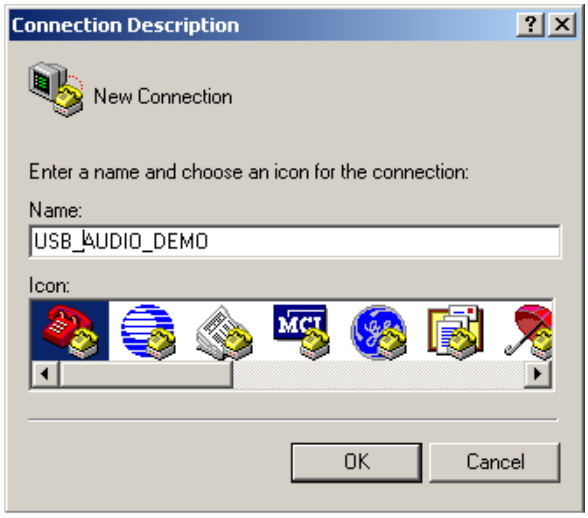


Figure G-9. HyperTerminal startup

c) After selecting the COM port, configure baud rate and other properties as shown in Figure G-10 (Baud rate: 115200, Data bits: 8, Parity: None, Stop bits: 1, Flow control: None).

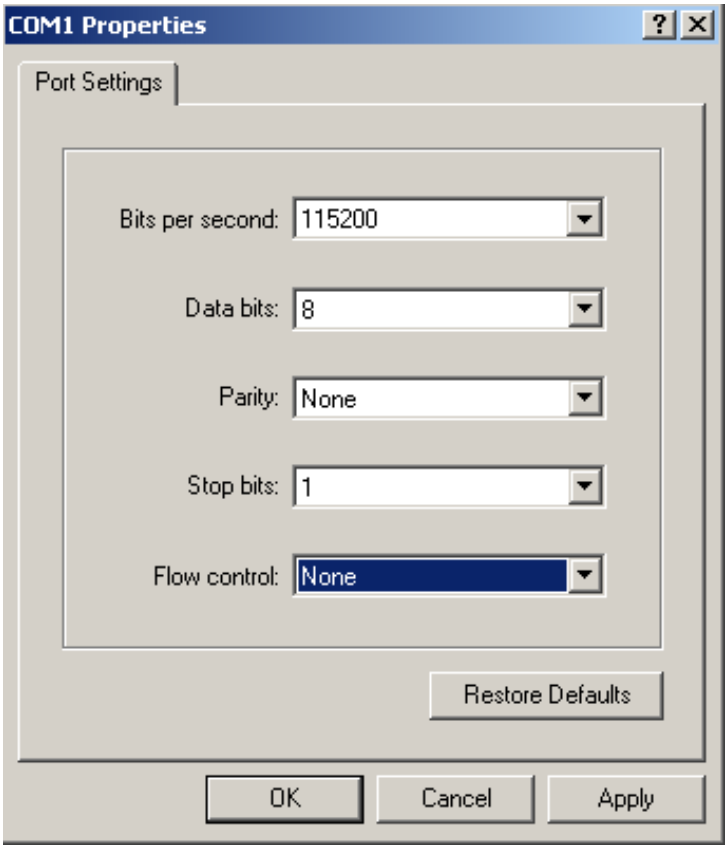


Figure G-10. COM properties

d) The HyperTerminal is configured now as shown in Figure G-11.

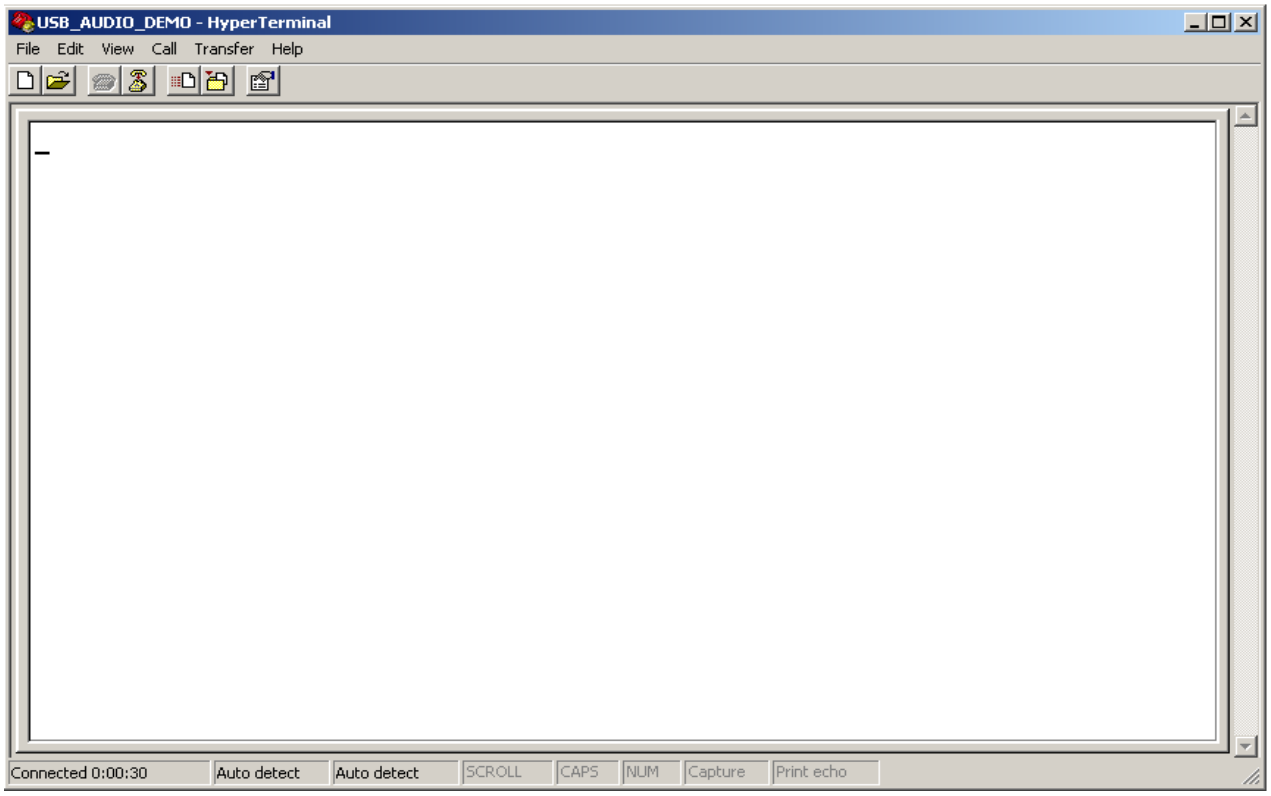


Figure G-11. HyperTerminal

e) Adjust master volume (volume, on/off/ Mute); the changed values are displayed on the HyperTerminal screen (Figure G-12).

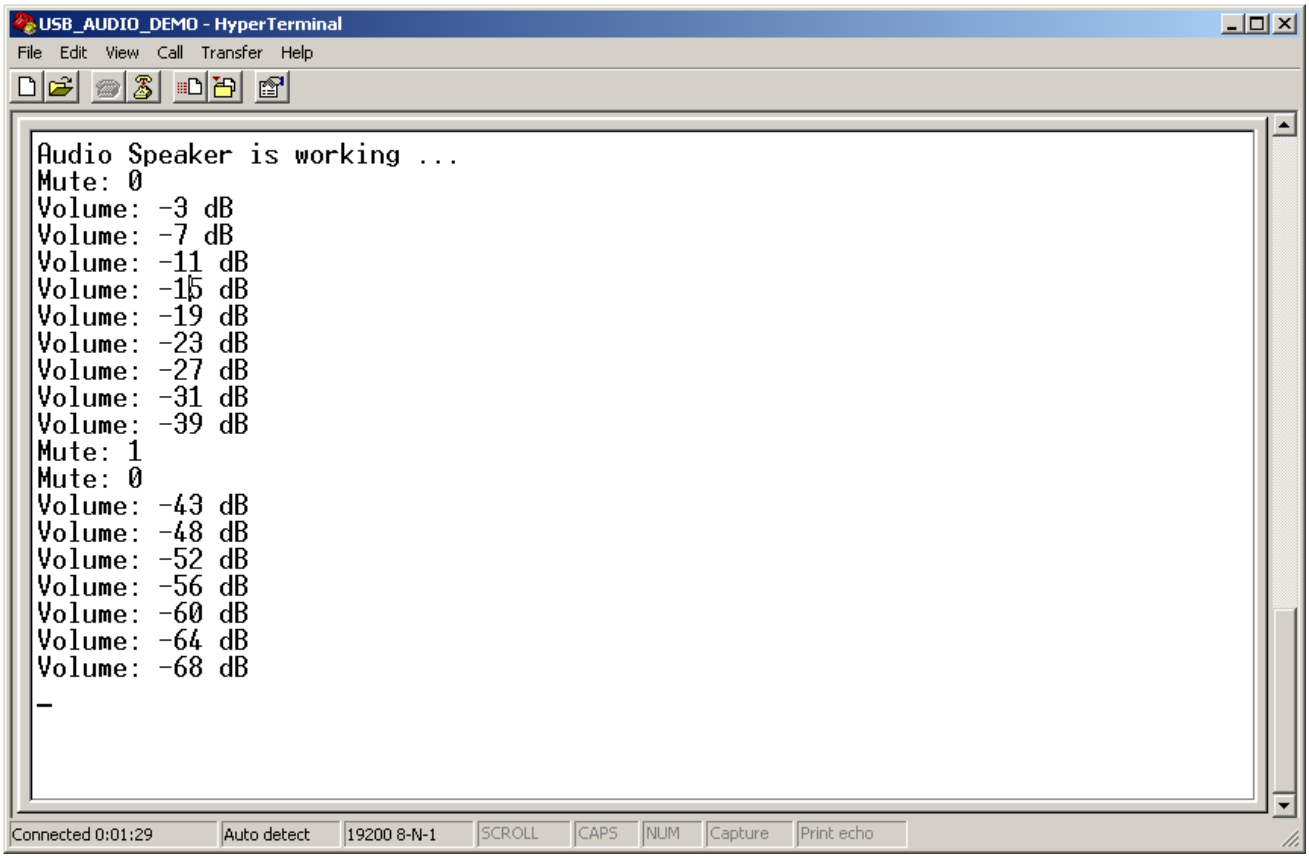


Figure G-12. Volume and mute control

f) Open the Window Media Player application Figure G-13 and then select and listen to your favorite audio; you can hear the song clearly.

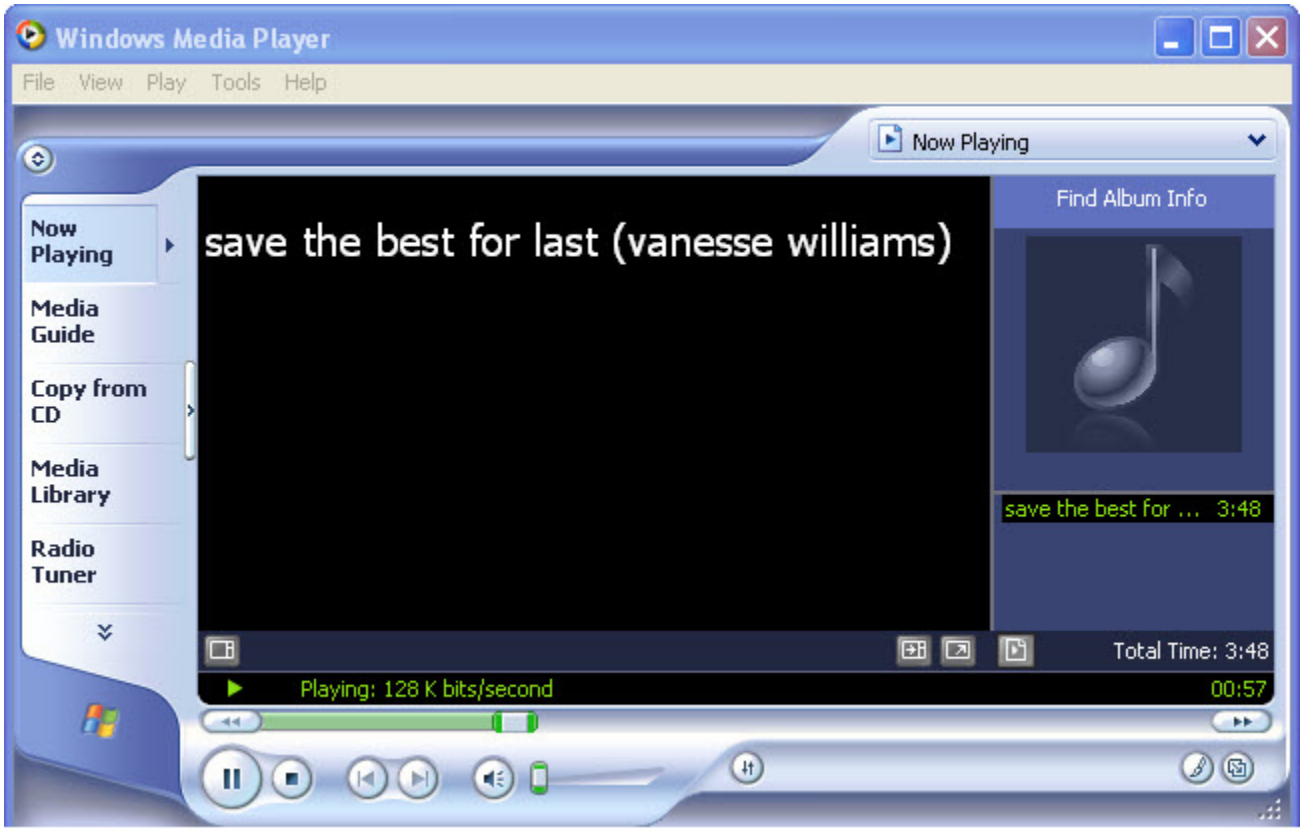


Figure G-13. Window Media Player

G.2 Audio generator demo

G.2.1 Setting up the demo

Figure G-1 describes the demo setup. DemoJM is connected to a PC using two USB cables and one speaker through an external circuit. The PC uses one USB cable to supply power to the board and program the image to the flash. The PC also acts as the host system and the DemoJM board acts as the USB device. They are connected by the second cable. A COM connection is also established to display the log of DemoJM.

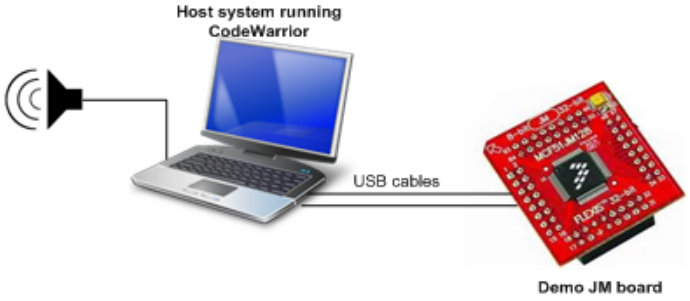


Figure G-14. Audio speaker demo setup

An audio data file (.wav file) which was converted to data arrays using the audio reproduction technique is stored in the device memory to transfer to the host.

G.2.2 Running the demo

1. Follow the steps from 1 to 8 in Section G.1.2, "Running the demo" to completely install the demo. Once installed, the demo can adjust the volume and mute.
2. Open the Sound Recorder application as shown in Figure G-15.

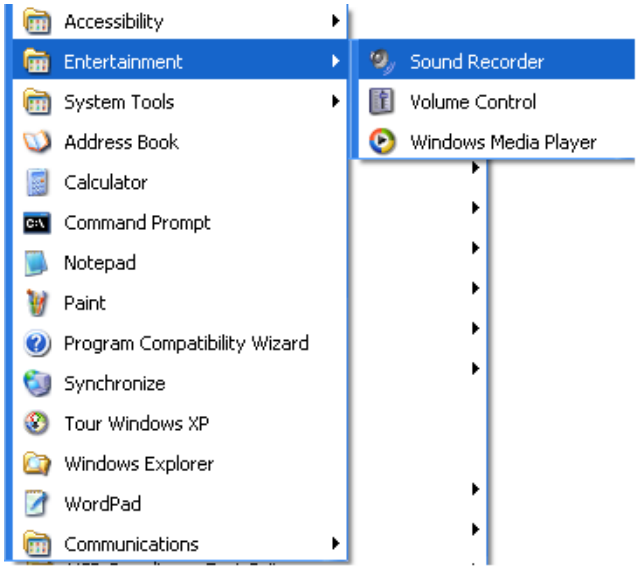


Figure G-15. Launch Sound Recorder application

3. Turn on demo board. The data of sound stored in the DemoJM memory will be sent to the host. It acts as a generator sending audio data to the host.

4. The sound is recorded by sound recorder application as shown in Figure G-16.

Figure G-16. Sound recorder

5. After recording, click on the Play button to listen the recorded sound. The sound that you can listen now is identical to the instance sound located in the memory.

Appendix H DFU Class Demo

This chapter explains how to use DFU Class Demo. It illustrates features of DFU Class and the Demo consists of two applications:

- **DFU device application**—An application developed based on the DFU class.
- **DFU PC host application**—A PC application developed to support download and upload firmware.

Download and upload processes are implemented through specific requests. The details of these requests are described in the DFU Class Specification. To take you through this chapter, the DFU device demo is illustrated by using a Demo JM board (DemoJM128).

H.1 Setting up the demo

Figure H-1 describes the Demo setup. The DemoJM board is connected to a PC (host) using two USB cables. One USB cable is used for powering the board and download the software image via BDM and must be connected between PC and USB P&E multilink port (J1000). The second cable must be conected between the mini AB port (J9) and PC which is considered HOST. The J11 jumper on the DEMOJM board must be set on the OFF position (in this case the board is DEVICE)

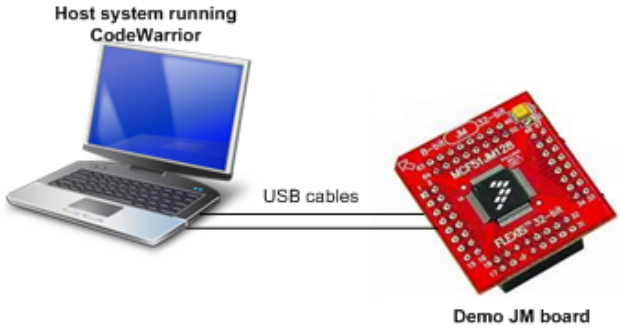


Figure H-1. DFU class demo setup

H.2 Running the demo

H.2.1 Driver installation

After the system has been set, you must follow these steps to run the demo:

1. Turn on DemoJM board, as soon as you turn on the device, it is recognized by the host and a callout as Figure H-2 appears on the right bottom corner of your screen. A Found New Hardware Window appears as shown in Figure H-3.

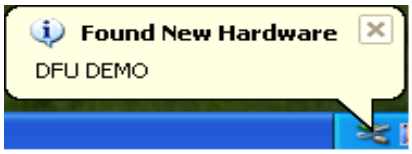


Figure H-2. Find New Hardware callout

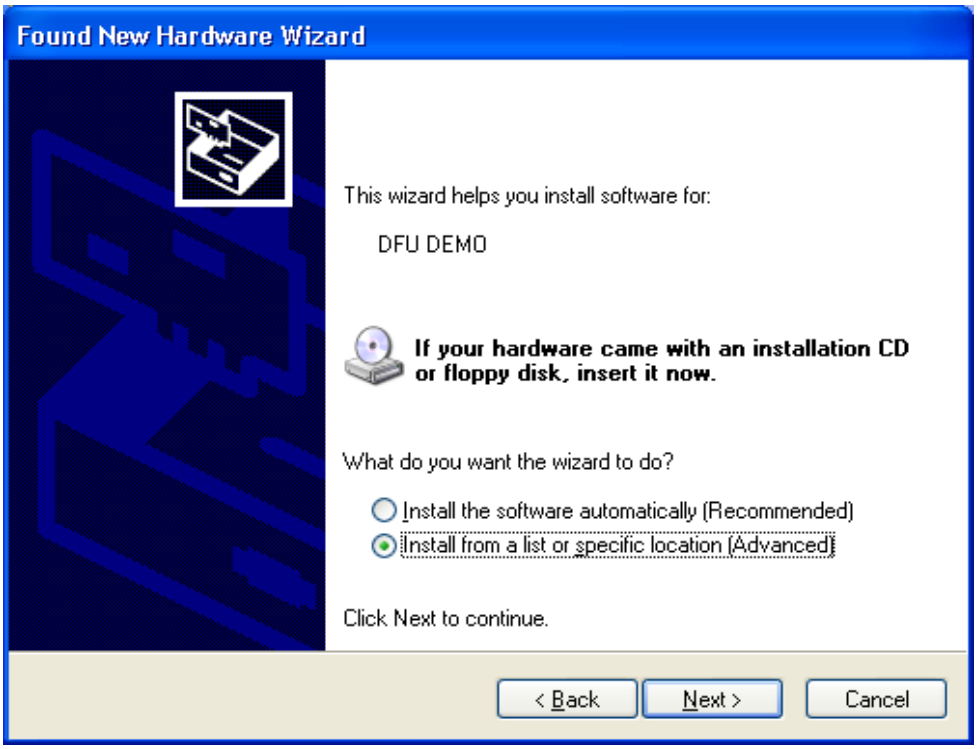


Figure H-3. Found New Hardware

2. Select Install from a list or specific location (Advanced) option and click on the **Next** button. Search and installation options window appears as shown in Figure H-4. Select **"Don't search, I will choose the driver to install"** option and click **Next**.

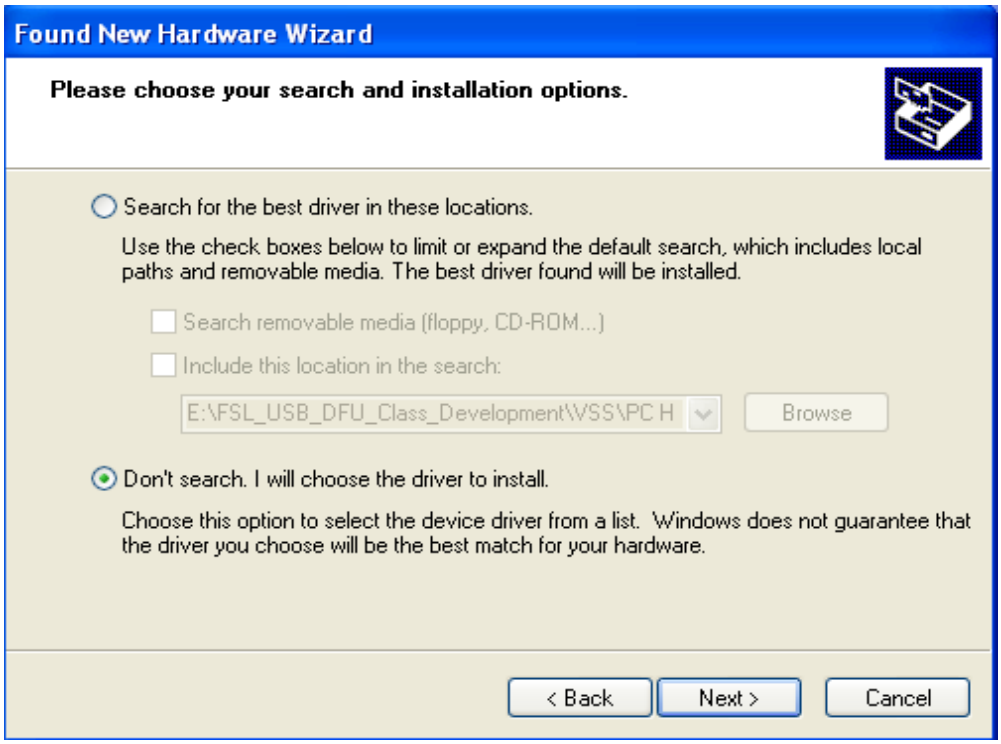


Figure H-4. Search and installation options

3. Hardware Type Window appears, Select **Show All Devices** option, and click **Next** button. Click **Have Disk** button when Select device driver window appears (Figure H-6).

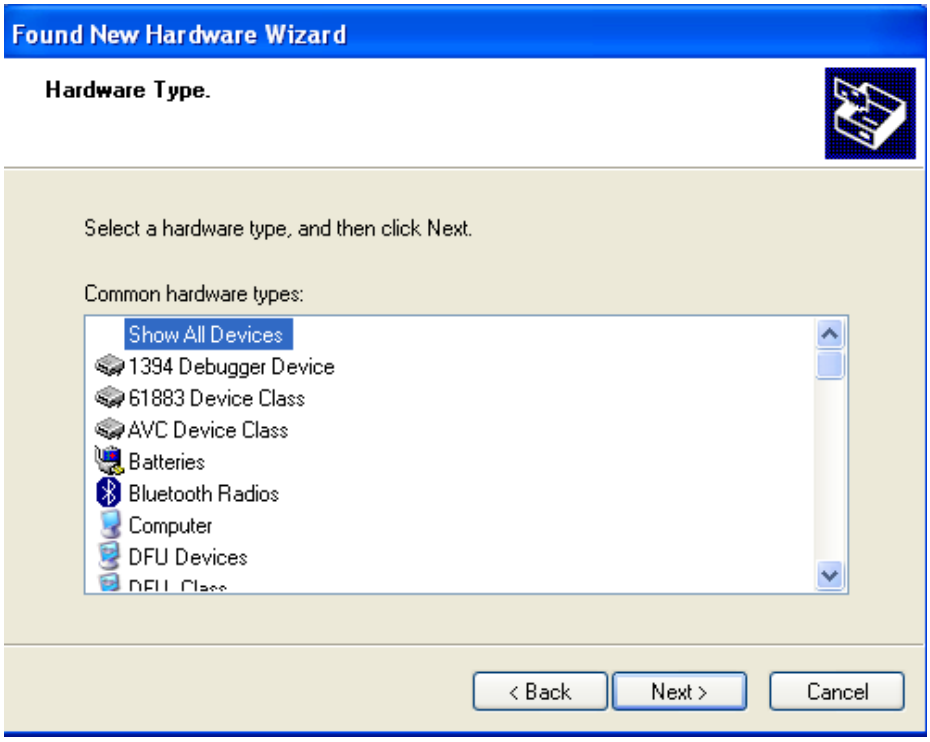


Figure H-5. Hardware type window

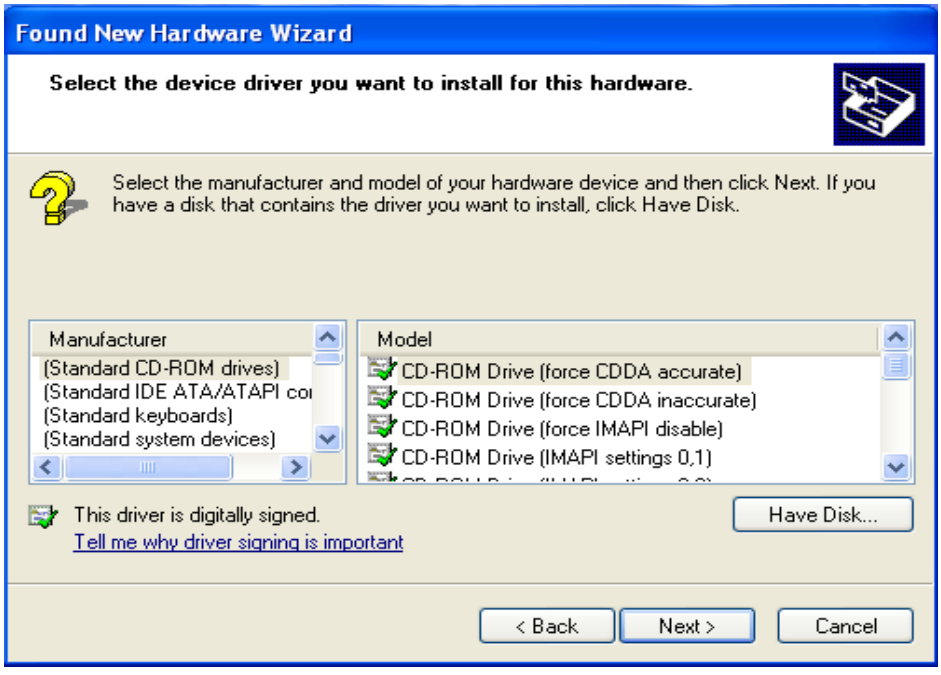


Figure H-6. Select device driver window

4. Navigate to the INF file location to choose an .INF file and click **Open** (Figure H-7). After that click **Next** when the next window appears to install driver.

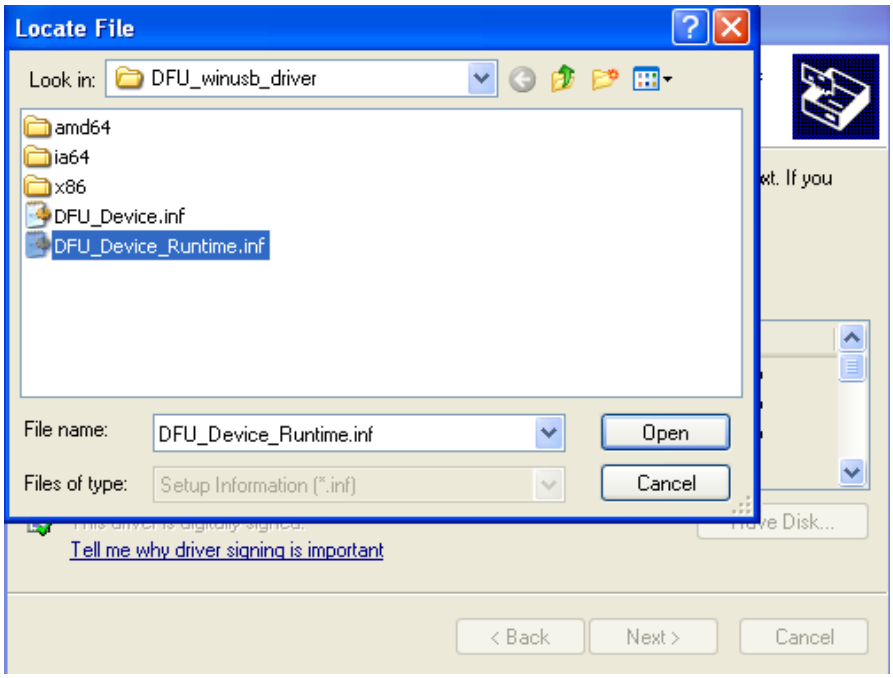


Figure H-7. Location to the driver

Once the driver is installed, Windows now recognizes it is a DFU device. But now it is in run-time mode so that HID mouse application is running on it.

To verify the installation, open the Device manager. You will see the Device firmware upgrade (DFU) and USB Human Interface Device entry (Figure H-8).

Figure H-8. DFU Device and Human Interface device in Device Manager

5. Launch DFU PC host application. The PC host application also recognizes RUNTIME mode is running as shown in Figure H-9. Click “Enter DFU mode” to switch the device to DFU mode.

‘

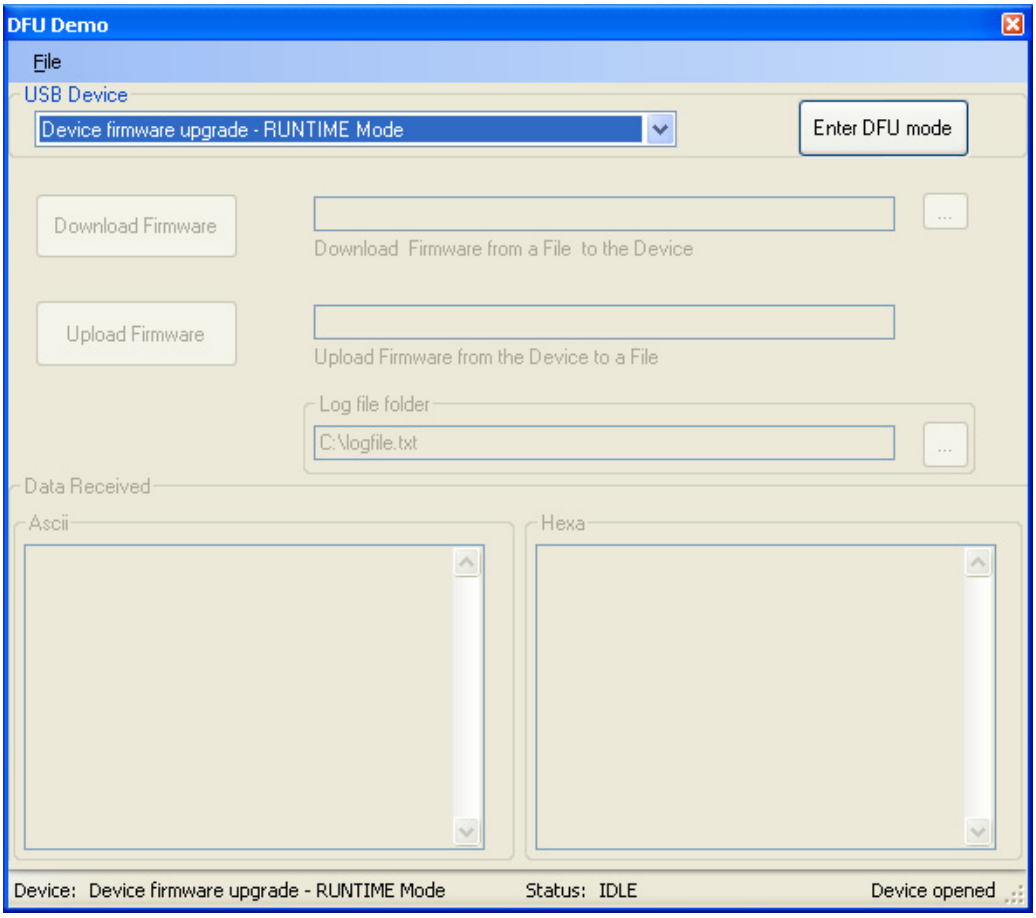


Figure H-9. Device firmware upgrade - Runtime mode

Once DFU mode is entered, the windows will ask for driver again. Follow Step 2 to Step 4 to install driver again (but driver for DFU mode as shown in Figure H-10).

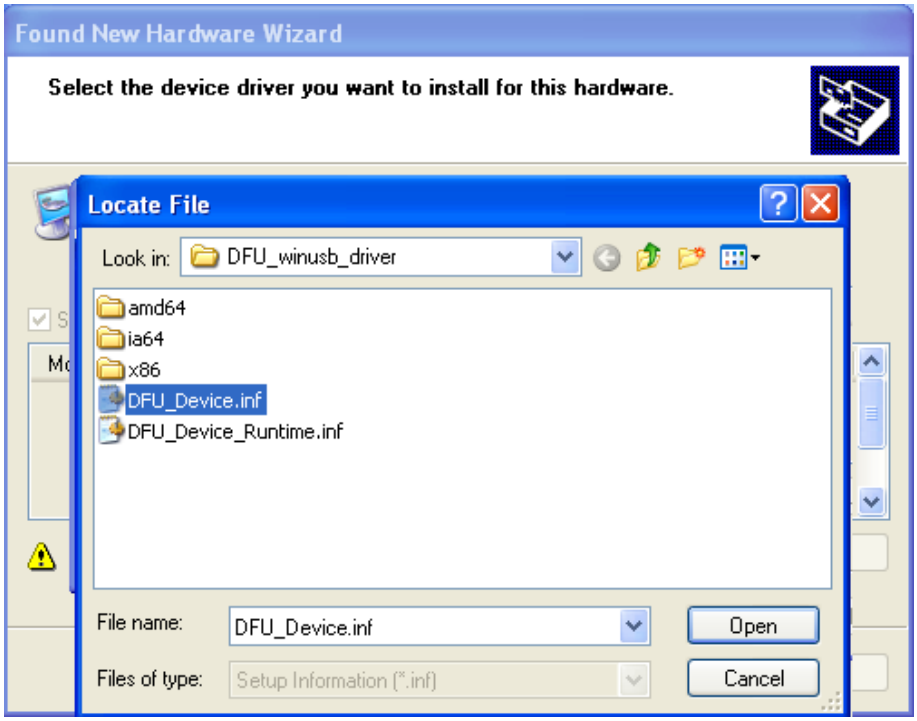


Figure H-10. Install driver for DFU mode

Once driver for DFU mode is been installed successfully, DFU device demo is in DFU mode and ready to use (Figure H-11).

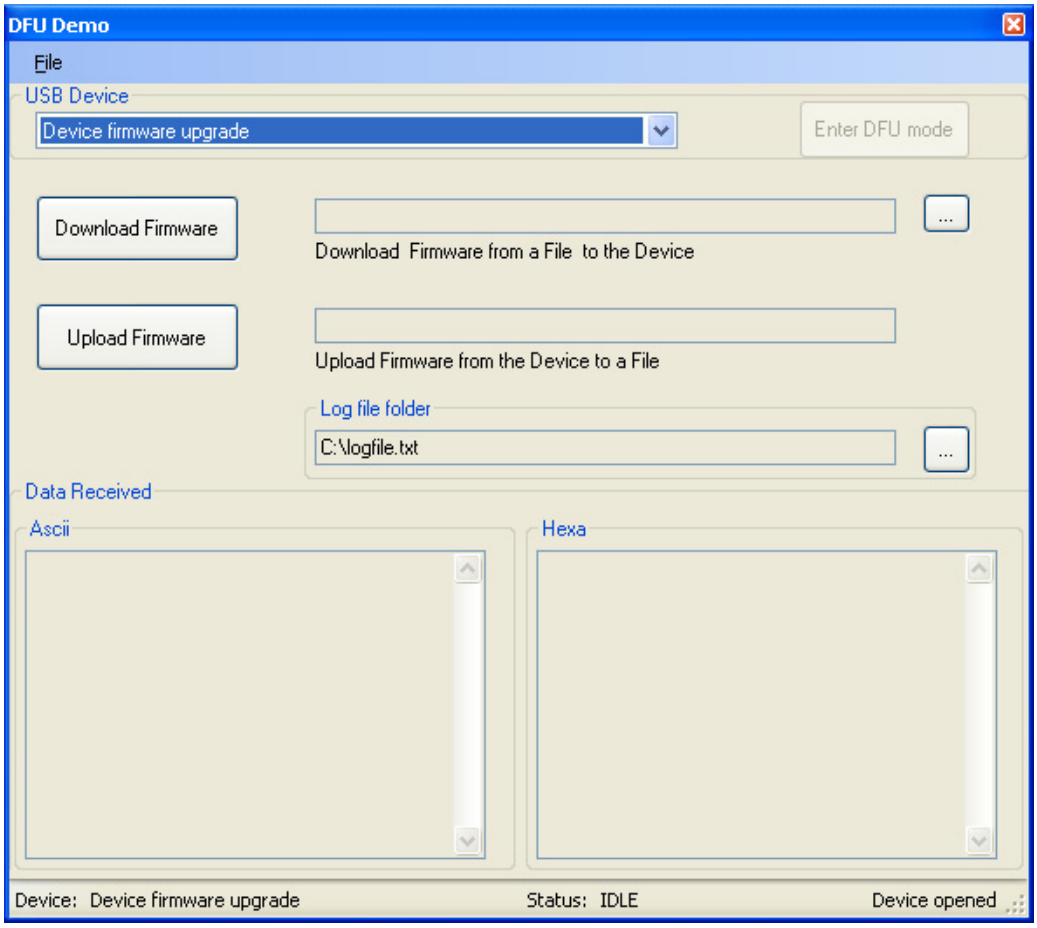


Figure H-11. DFU device demo in DFU mode

H.2.2 Downloading firmware

When the driver is installed completely, choose a firmware file to download to the device as shown in Figure H-12.

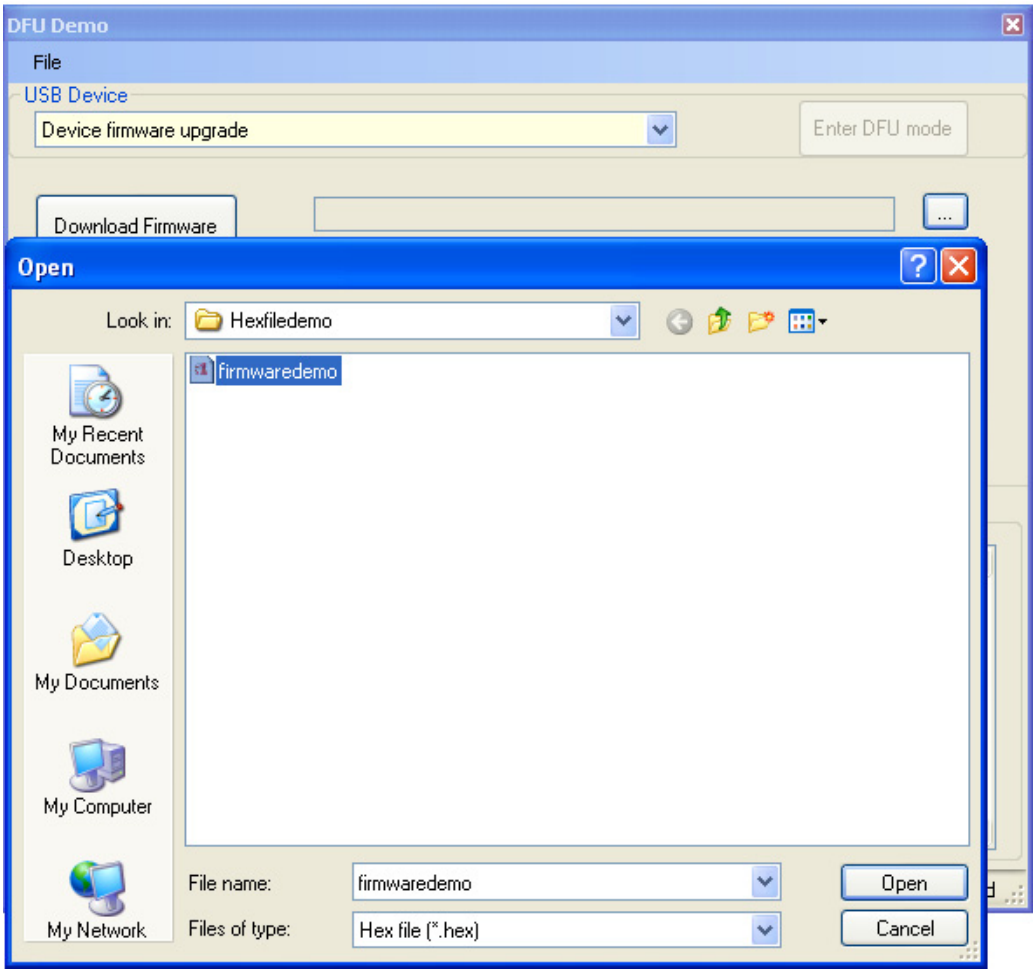


Figure H-12. Choosing the firmware file

The content of the firmware file will be displayed in ASCII and HEX as shown below (Figure H-13).

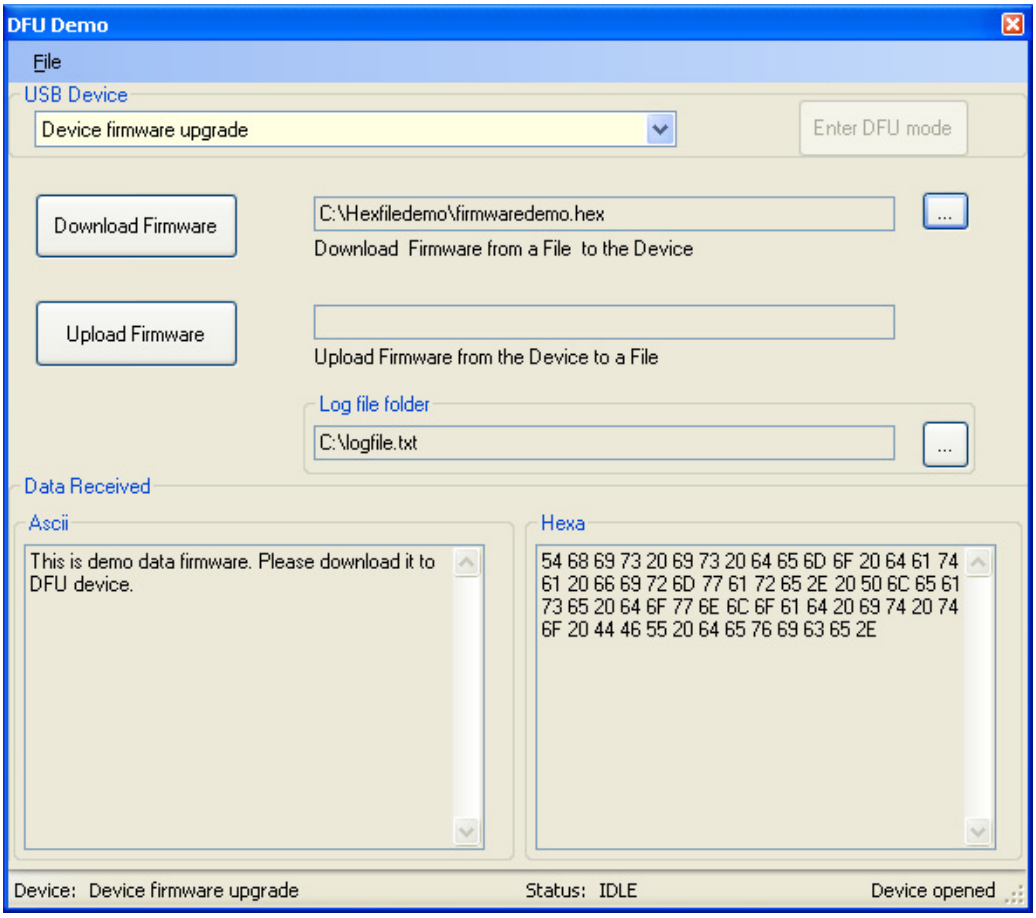


Figure H-13. Firmware contents

Click **Download Firmware** button, the firmware will be downloaded to the device (Figure H-14).

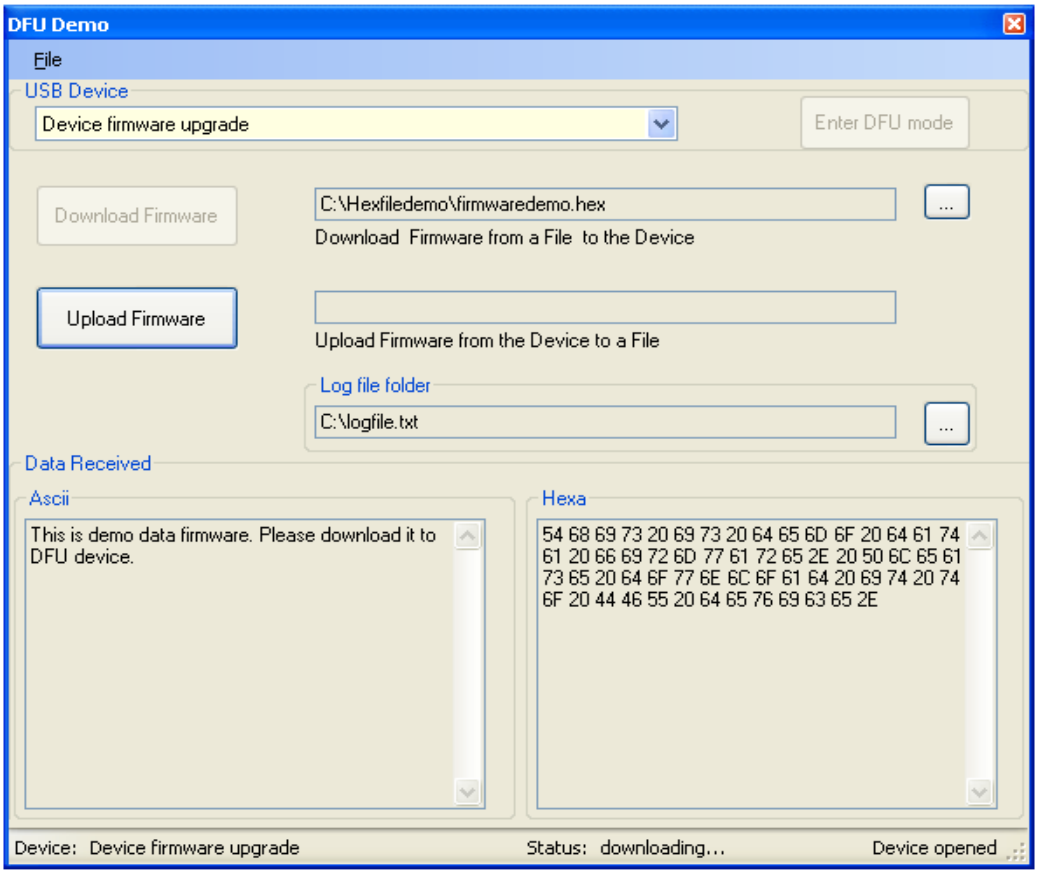


Figure H-14. The firmware is ready to download

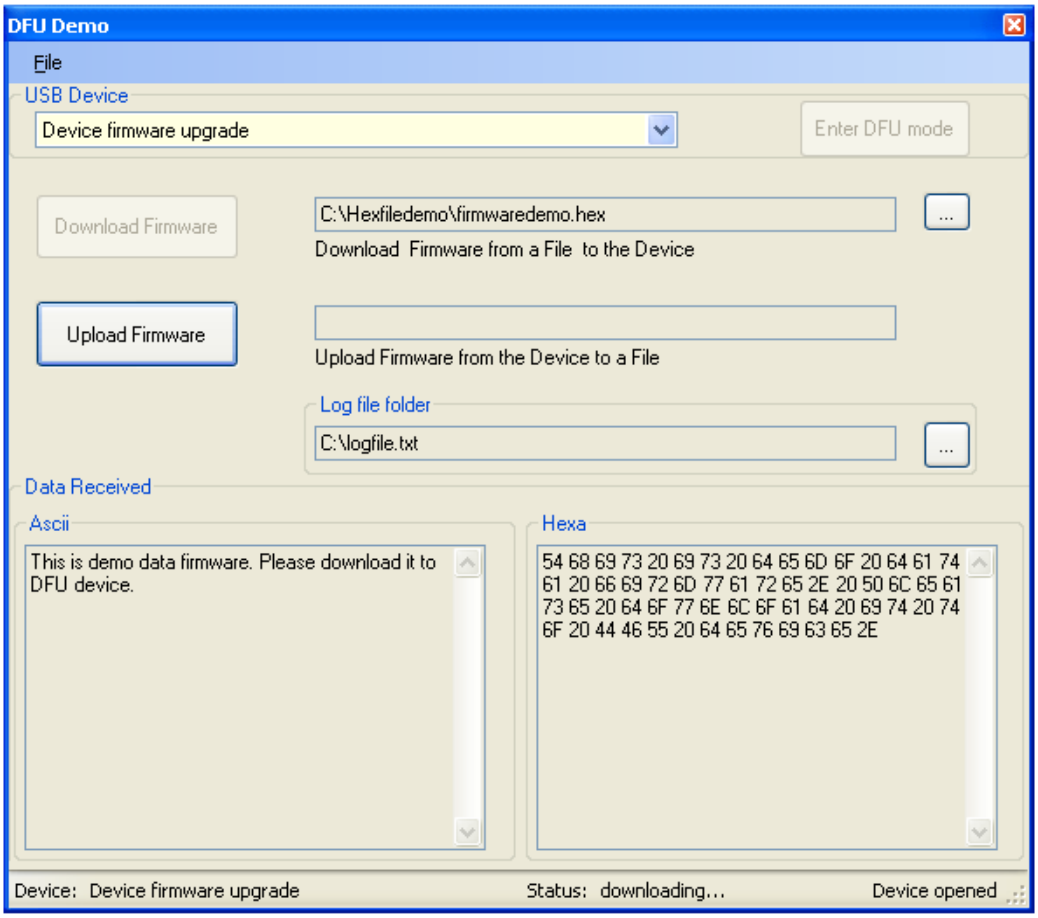


Figure H-15. Firmware is downloading

Once the download firmware process is finished, the PC host informs to you that the firmware was updated successfully as given in (Figure H-16).

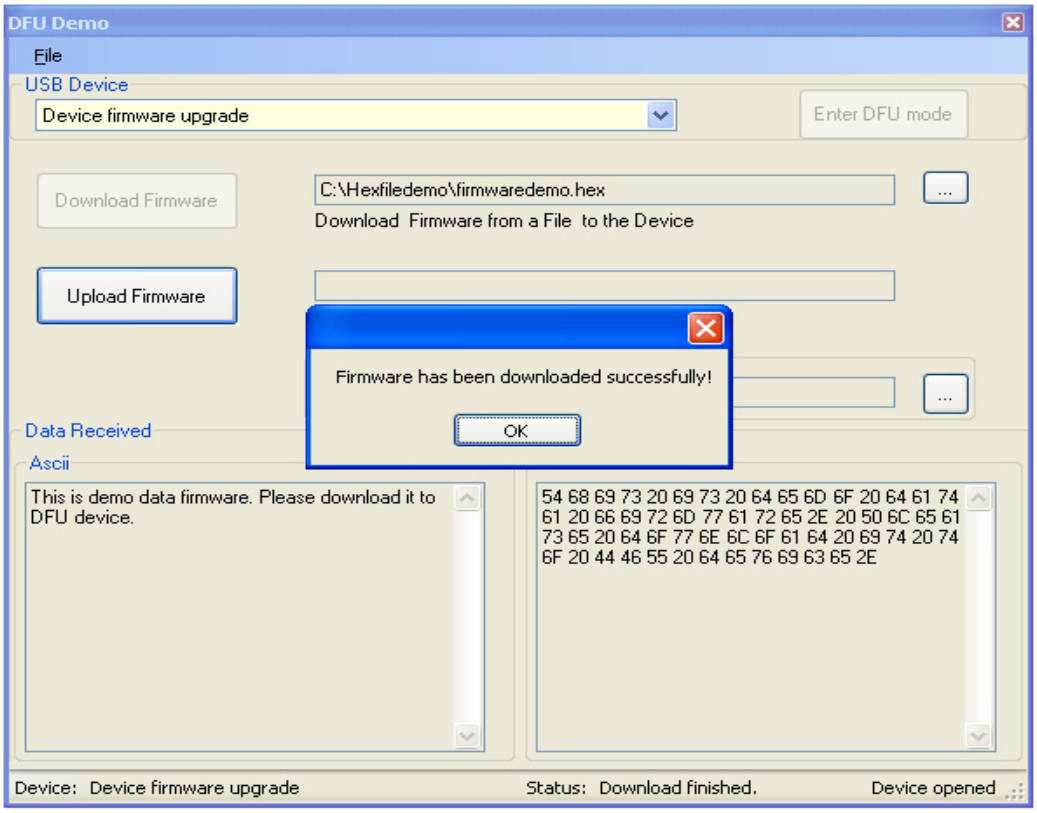


Figure H-16. Download is complete

To verify states of the device while in download firmware process:

- Open the log file in **C:** (this is default location for log file, you also can choose another place to store log file by clicking the choose log file path button).

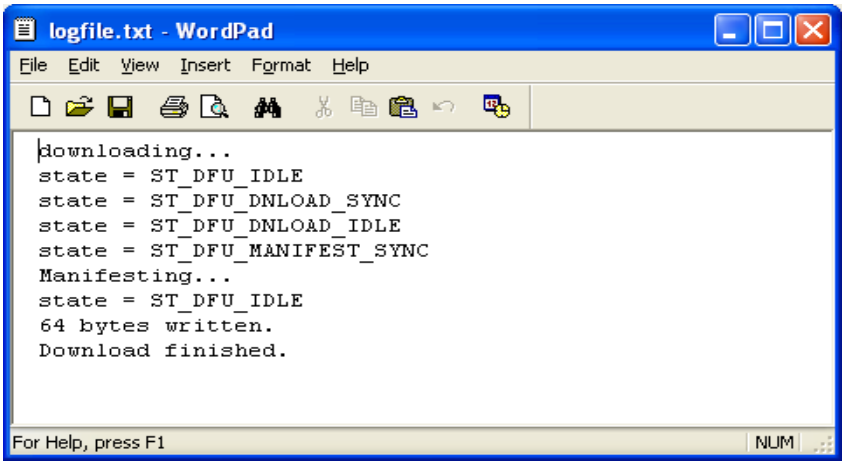


Figure H-17. Content of log file

NOTE

If the USB cable is unplugged during the download process, the DFU PC host will ask to continue download whenever the USB cable is plugged, again (Figure H-18).

.

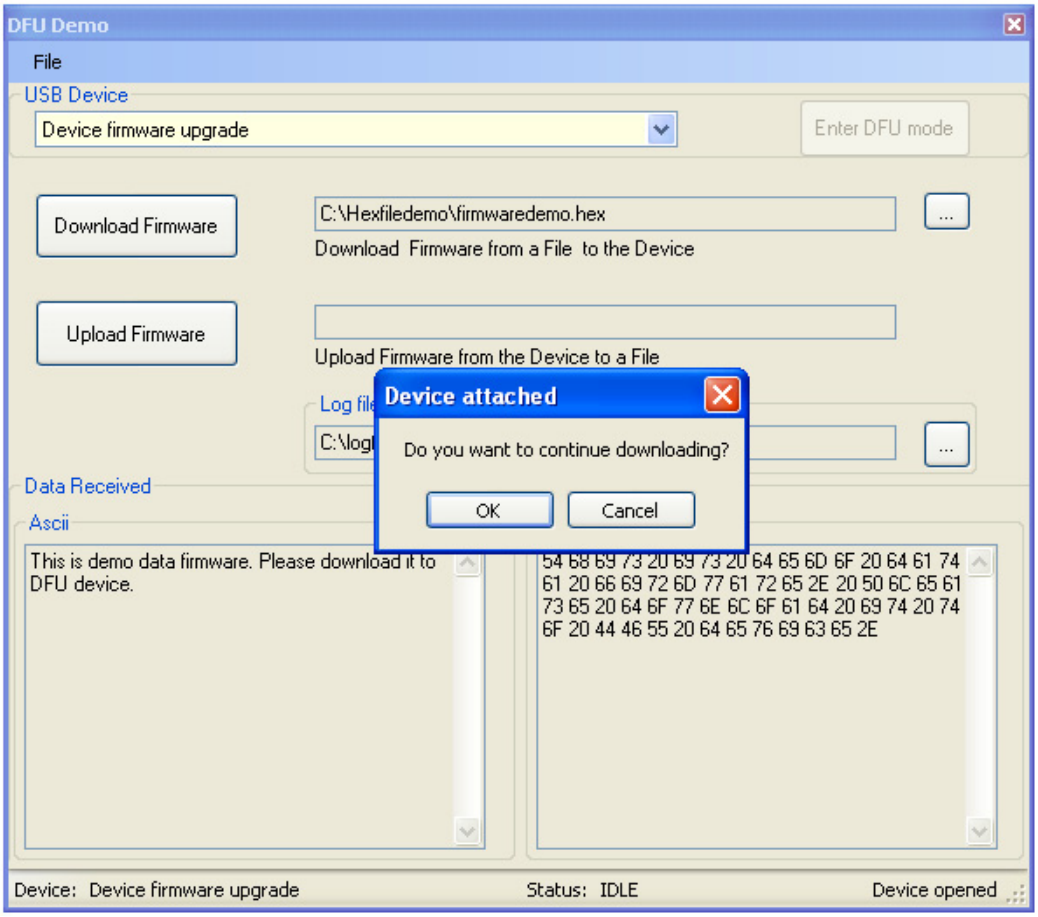


Figure H-18. Resume downloading

H.2.3 Upload firmware

Whenever you want to upload firmware, click Upload firmware button on DFU PC host application. The application will ask for file name, type the file name and click Save button. After that device firmware will be uploaded, immediately.

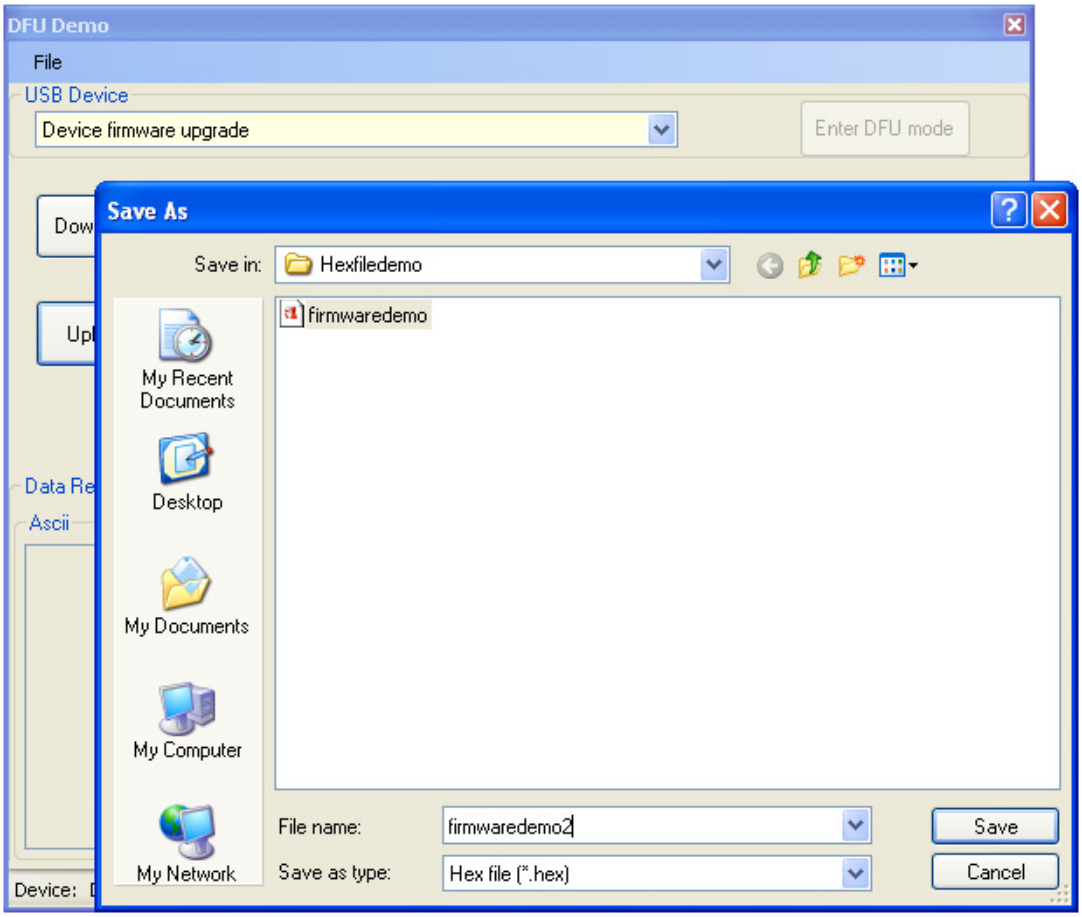


Figure H-19. Save the uploaded firmware file

When the upload process is finished, the data received is displayed in two text boxes because the upload function reads total 256 bytes of device's firmware, so some bytes in flash memory which that were not written be read like 0xFF.

Appendix I Battery Charging Device Demo Application

I.1 Setting up the demo

Figure I.1 describes a typical connection for battery charging demo application.

Figure I-1. Battery charging demo setup

The target board is a modular Tower system composed of two elevators (primary board elevator used for power supply with the second which is a dummy board used for mechanical stability), one serial board (TWR-SER) handling the communication over the RS232 and/or USB and the microcontroller module (eg: TWR-K40N512 for a K40 processor, TWR-K60N512 for a K60 processor a.s.o.)

The Host PC is running the Code Warrior for build, download and debug purpose; the application functionality can be monitored through a serial terminal such as the Microsoft Windows HyperTerminal. The Host PC is connected via a RS232 cable (3-pin null modem type) to the serial port of the TWR-SER board.

The mini-AB port of the TWR-SER from the Target Board may be connected to either a PC type A USB port, or to a wall USB charger.

The jumper setting for the serial board (TWR-SER) of the Target board shall be as below:

- USB settings:
  - J16 (USB Mode Select): 5-6 OTG Mode (need the OTG mode for VBUS detection)
  - J11 USB OTG Interrupt select (1-2: IRQ_H, 3-4: IRQ_F, 5-6: IRQ_D, 7-8: IRQ_B)

— J10 (USB VBUS Select): 2-3 BUS powered device (source 5V from USB)

- Serial communication settings:
  — J15 (RS232/RS485 Select): 1-2 RS232
  — J19 (Tx Select): 1-2
  — J17 (Rx Select): 1-2

The Figure H-2 shows the location of each header mentioned above, for an easily identification on the communication board.

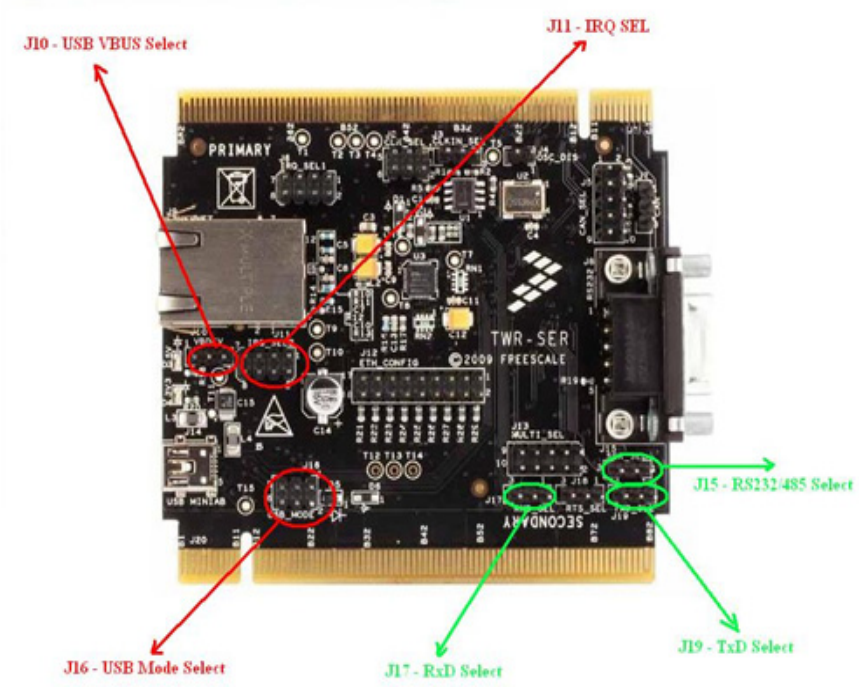


Figure I-2. Battery cahrging related board jumpers

I.2 Running the demo

For the battery charging class, the demo application is located in **\Device\app\batt_chg** path. Perform the following steps to run the demo application:

1. Open the project and load the images on the flash of the board controller.
2. Open a serial terminal performing the setting steps already presented; starting the demo application, and with USB cable left unconnected, the terminal shows a message requesting the user to connect the device to a charging port.

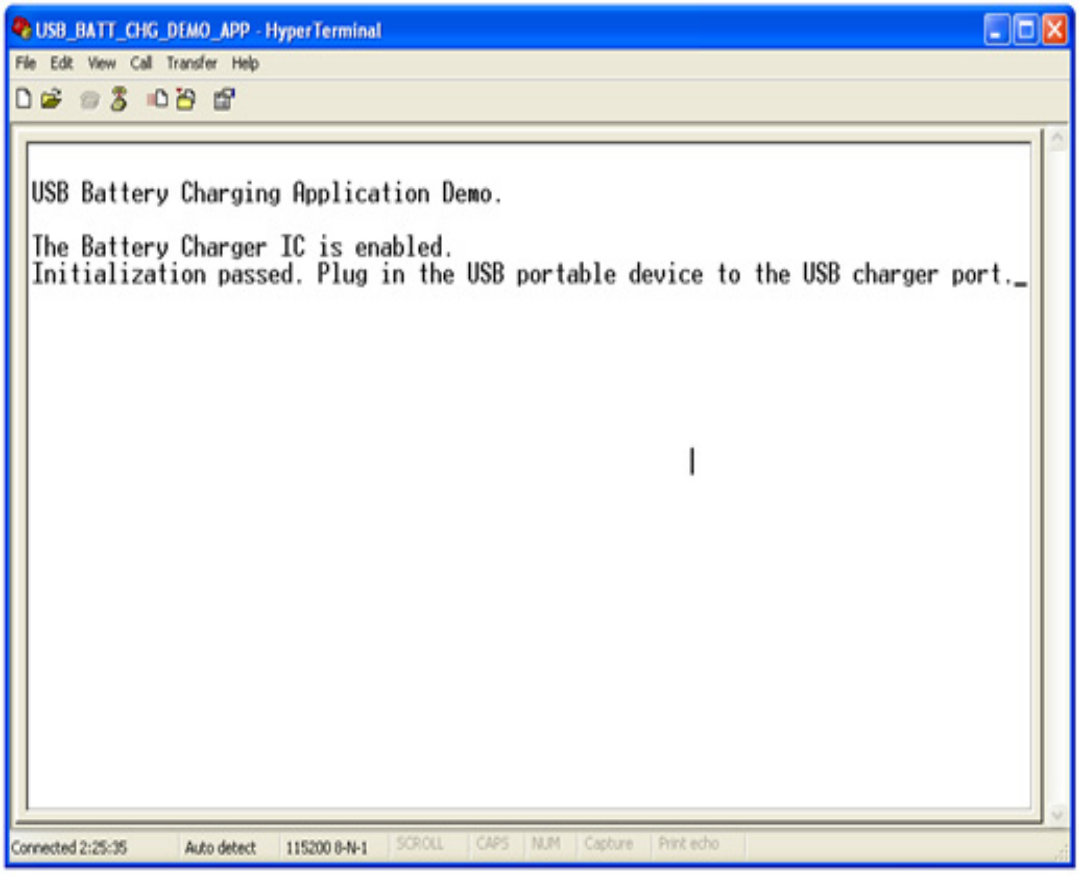


Figure I-3. USB Battery Charging Demo App – after initialization

3. Connect the USB cable to a PC; in this case the application detects that it has been connected to a Standard Downstream Port (SDP) and it also indicates the maximum allowable current to be used for battery charging process, as it is shown in the next screenshot.

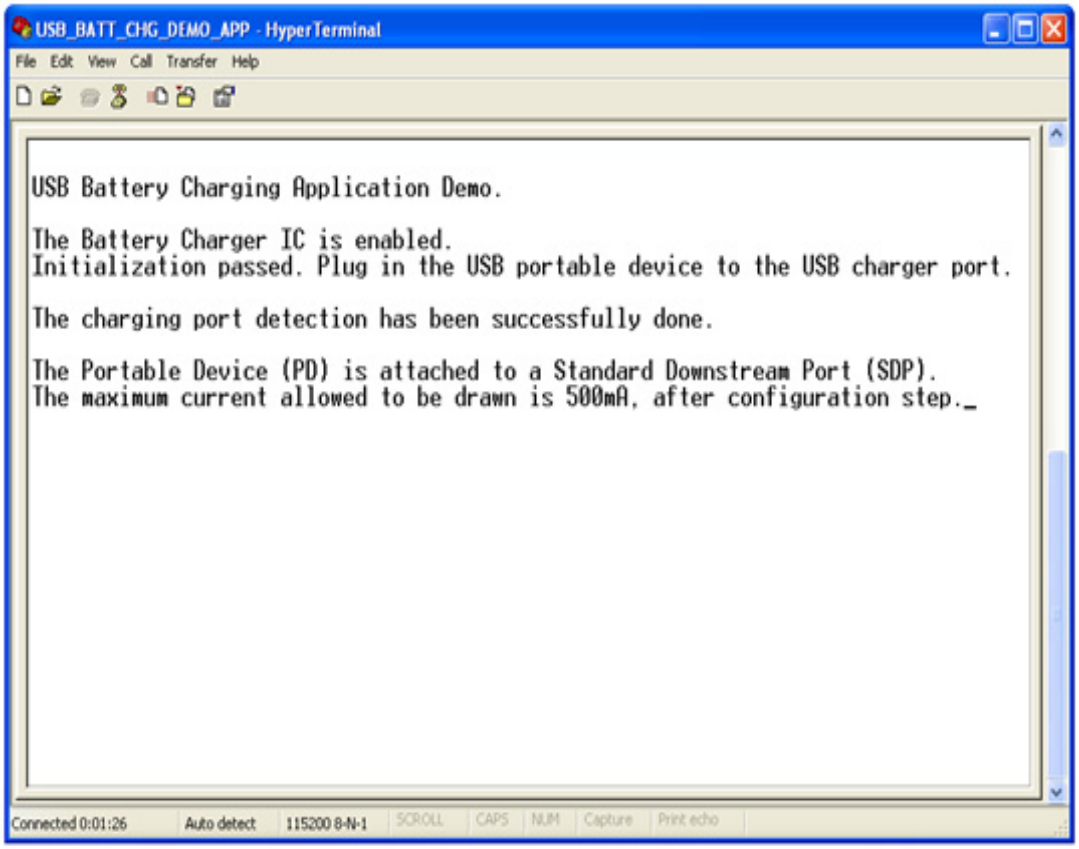


Figure I-4. USB Battery Charging Demo App – SDP connect detection

4. If instead, a wall charger via USB cable is connected to the target board the application recognizes this as a Dedicated Charging Port (DCP) type and the maximum allowable current is indicated as well.

NOTE

The wall USB Charger should be according to the Battery Charging Specification rev. 1.1, therefore the data lines D+ and D- should be shorted together via a resistance, RDCHG_DAT (less than 200 ohm) otherwise the USB port is not properly recognized as a DCP type.

Appendix J Video Device Class Demo Applications

J.1 Introduction

J.1.1 About Video Class demo

This section gives a quick overview on how to use the USB Video demo software package. The USB Video demo is developed base on USB Video Class. It demonstrates sending video data stream to the host like an USB Video Camera. It is called Virtual Camera application. The application also support specific requests from host such as Brightness control and so on.

To take you through this guide, the demo is illustrated by using a TWR-M5225X board.

J.2 USB Video Demo – Internal Flash

J.2.1 Overview

In this demo, the video data is stored in internal flash memory. The *Virtual camera* application will read video data from the memory and transfer it to computer through USB connection. Because of memory limitation, the video size is not too large. This application supports videos whose solutions are 176 x 144.

J.2.2 Setting up the demo

Figure J-1 describes the demo setup. TWR-M5225X board is connected to a PC using two USB. The computer uses one USB cable to supply power to the board and program the image to the flash. The computer also acts as the host system and the TWR-M5225X board acts as the USB device by using the second cable.

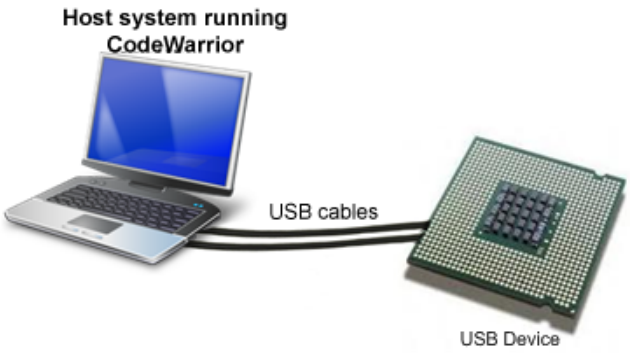


Figure J-1. Video demo setup

J.2.3 Running the demo

After the system has been set, you must follow these steps to run the demo:

J.2.3.1 Preparing

1. Run internal flash demo from folder *app\virtual_camera*. Video data is included in code and download to chip when running the application.
2. Plug USB Video device in to the PC (host). As soon as you turn on the device, it is recognized by the host and is installed automatically. You must see the callout as shown in on the right bottom corner of your screen as shown in Figure J-2.

Figure J-2. Find New Hardware callout

3. After successful installation, the host indicates that the device is ready to use as shown in Figure J-3

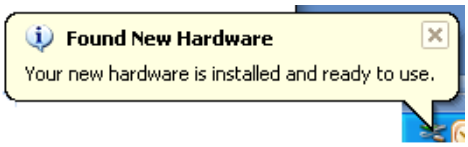


Figure J-3. Installation of USB Video device

4. To verify whether the USB Video Device has been installed properly or not, you must see the device entry in the device manager as shown in Figure J-4.

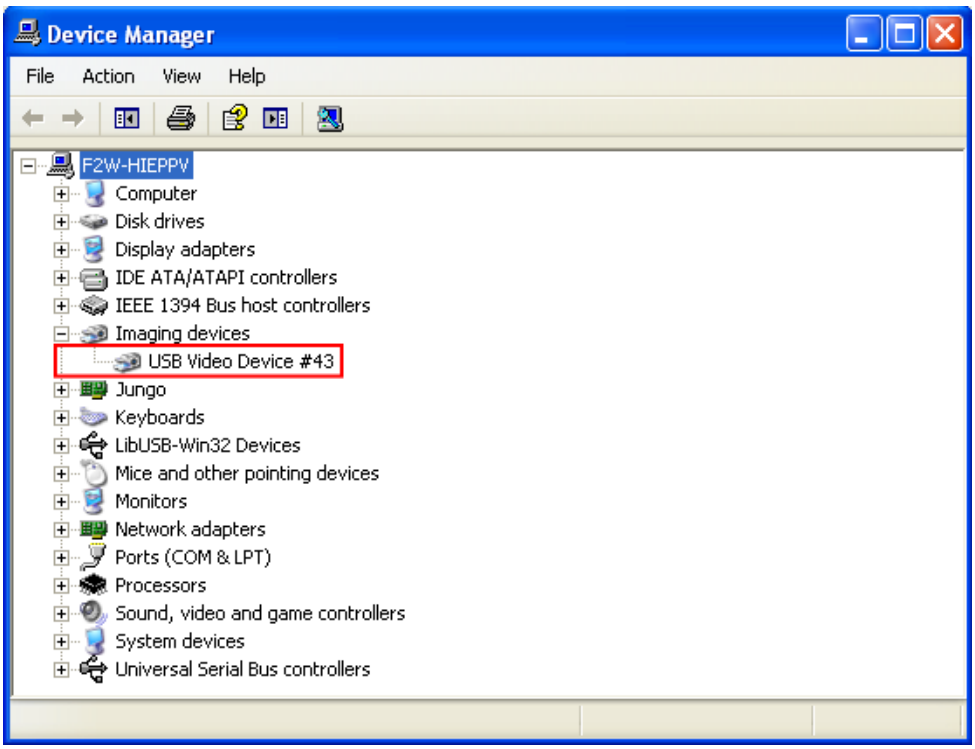


Figure J-4. Device manager dialog

5. Double-click on My Computer icon, USB Video device icon is also appeared as shown in Figure J-5

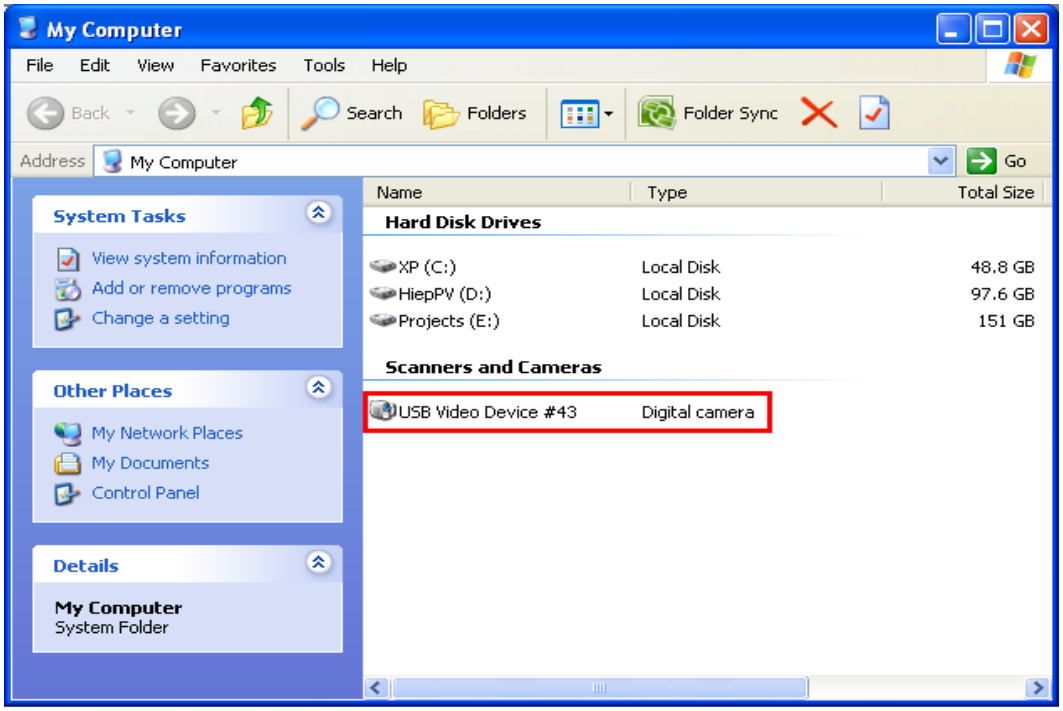


Figure J-5. USB video device entry

6. By Double-clicking on the USB Video Device icon, video is displayed as shown as shown in Figure J-6

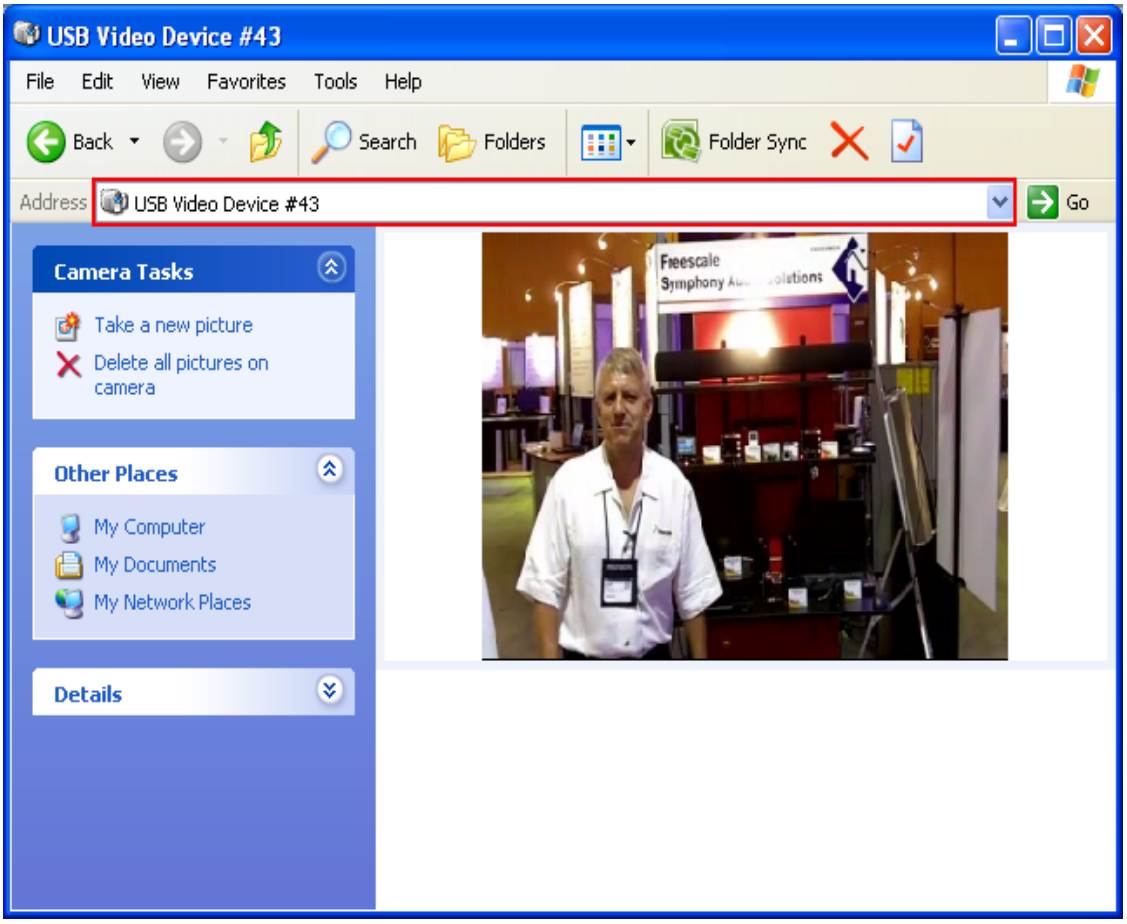


Figure J-6. Video is displayed

J.3 USB Video Demo — SD card

J.3.1 Overview

Internal flash memory isn't enough for big video, so we use a SDSC(Security Digital Standard Capacity card) to store video data. An application is called *sd_loader* is necessary to download video data from computer and write it into SDSC. The *Virtual camera SD* application will read video data from SDSC and transfer it to computer via USB connection. This application supports videos whose solutions are 320 x 240.

J.3.2 Setting up the demo

Figure J-7 describes the demo setup. TWR-M5225X board is connected to a PC using one USB cable. The cable is used by computer to supply power to the board. The computer also acts as the host system and the TWR-K60N512 board acts as the USB device by using the cable. You use PnE debugger to program

chip MK60N512VMD. Video data is downloaded from computer and stored in a SD card. Sections below will describe how to load a video to SD card by using sd_loader application and run video application.

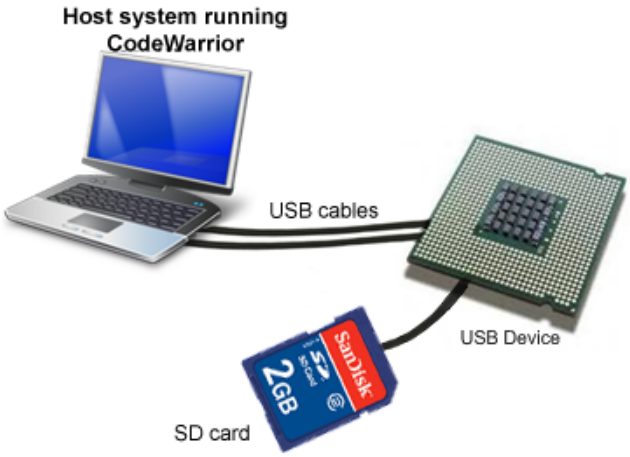


Figure J-7. Video demo setup

J.3.3 Running the demo

After the system has been set, you must follow these steps to run the demo:

J.3.4 Preparing video data

1. Running SD loader project from folder *app\sd_loader*.
2. Plug USB loader device in to the PC (host). As soon as, you turn on the device, it is recognized by the host and is installed automatically. You must see the callout as shown on the right bottom corner of your screen as shown in Figure J-8.

Figure J-8. Find new hardware callout

3. After successful installation, the host indicates that the device is ready to use as shown in Figure J-9.

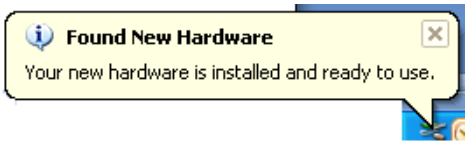


Figure J-9. Installation of USB video device

4. Double-click on My Computer icon, USB Video device icon is also appeared as shown in Figure J-10

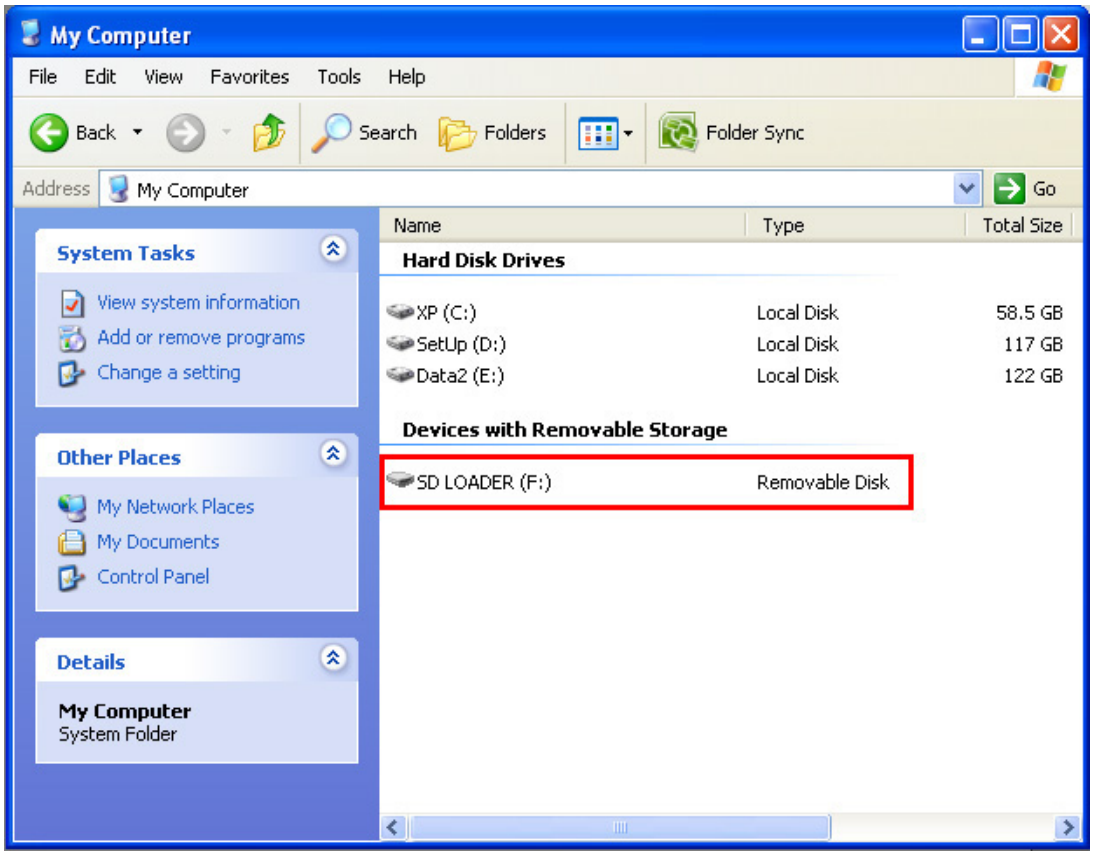


Figure J-10. SD loader

5. To verify whether the USB Video Device has been installed properly or not, you must see a "*READY.TXT*" file, after double-click on SD LOADER icon

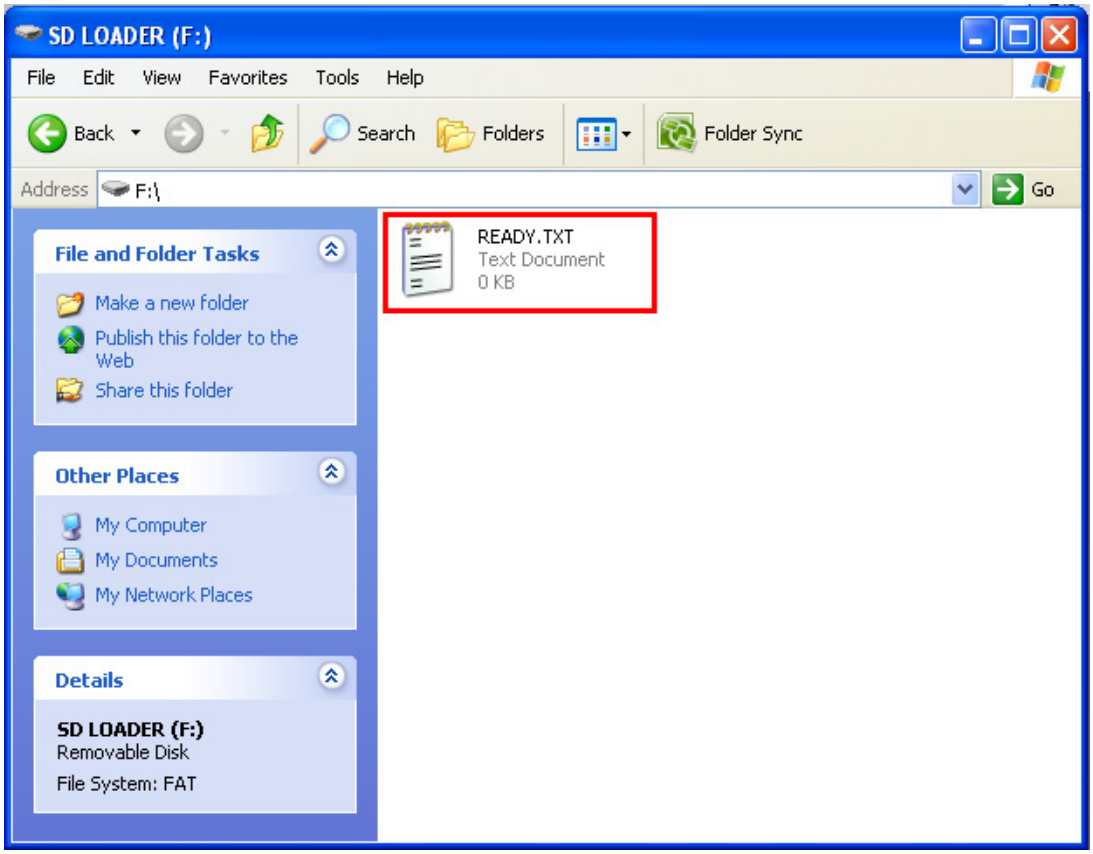


Figure J-11. Device ready status

6. Send your video to SD as following Figure J-12

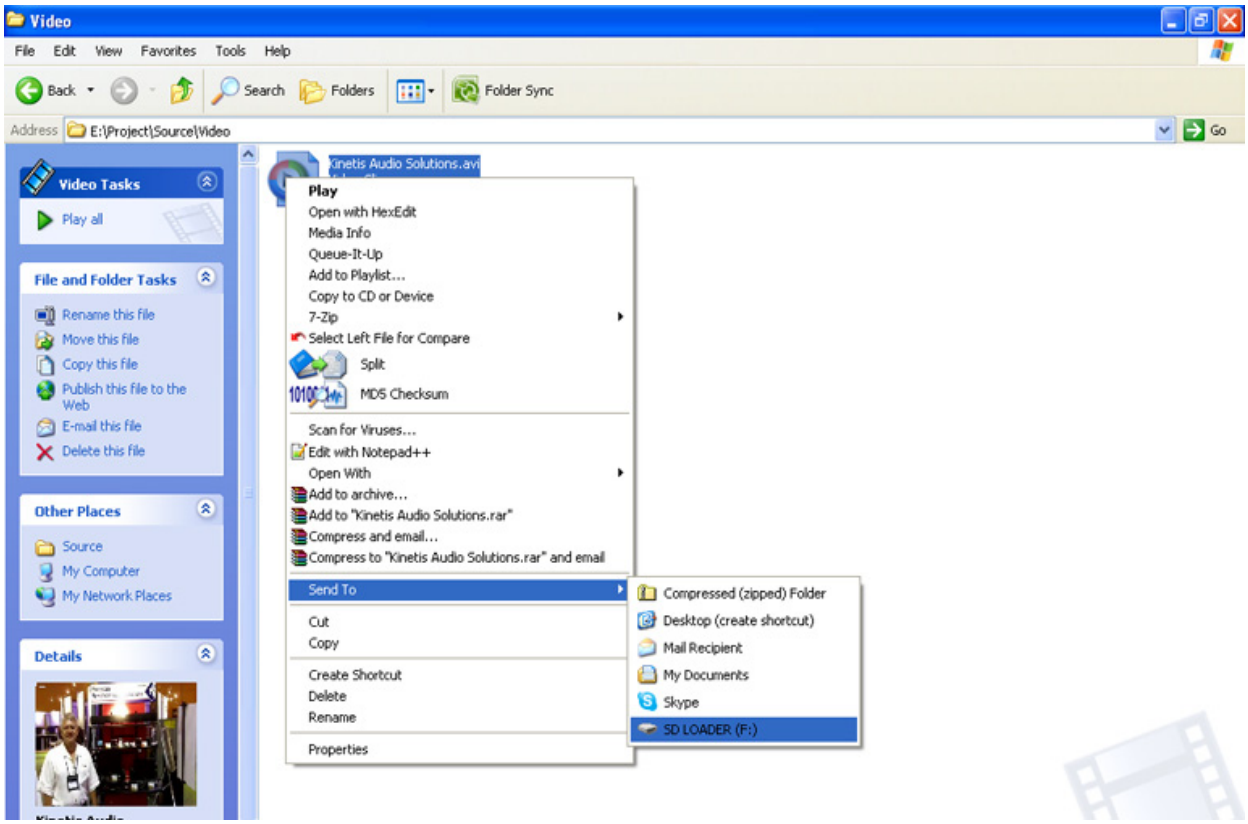


Figure J-12. Send video to SD loader

Then, waiting for transferring process

Figure J-13. Transfer data

7. When transferring finish, by Double-clicking on the SD LOADER icon, you will see a SUCCESS.TXT file Figure J-14.

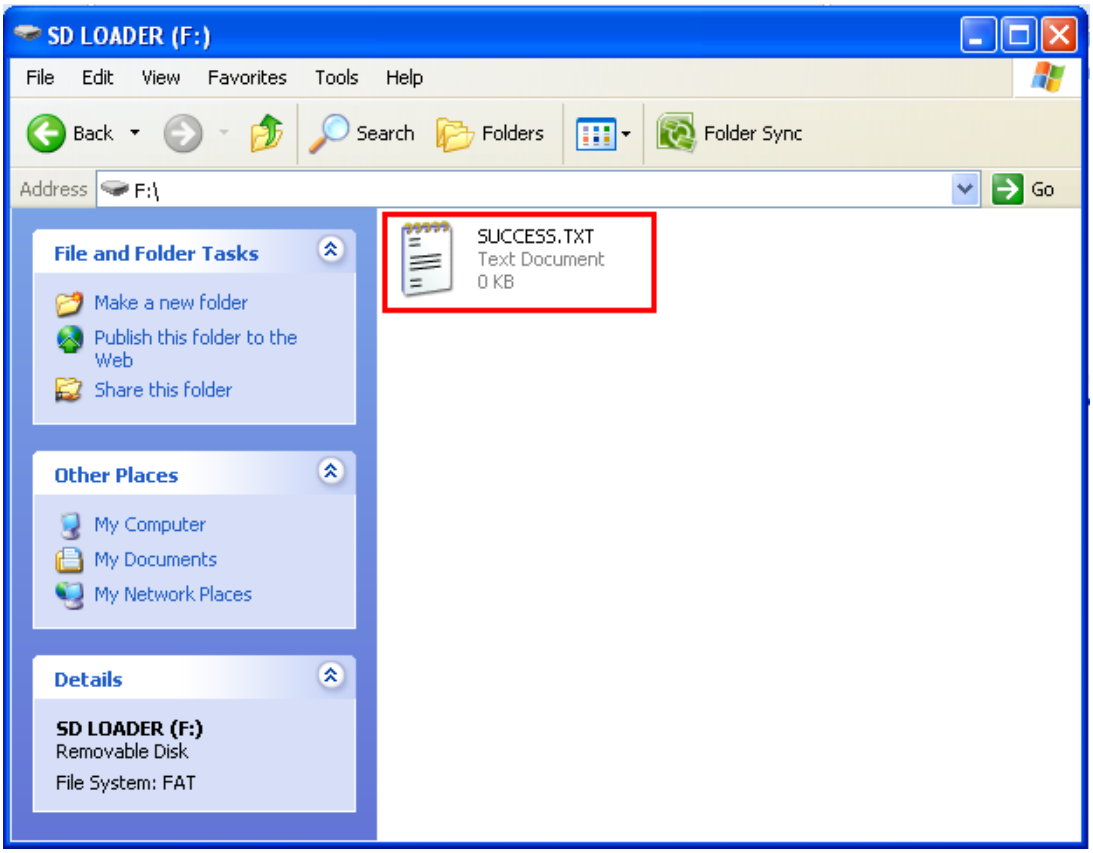


Figure J-14. Transfer success rate

J.3.4.1 Running video demo application

1. Running the Video_SD application from folder *app\Virtual_camera_SD*
2. You repeat steps from 2th to 7th in Internal flash demo, then video is displayed as shown as shown in Figure J-15

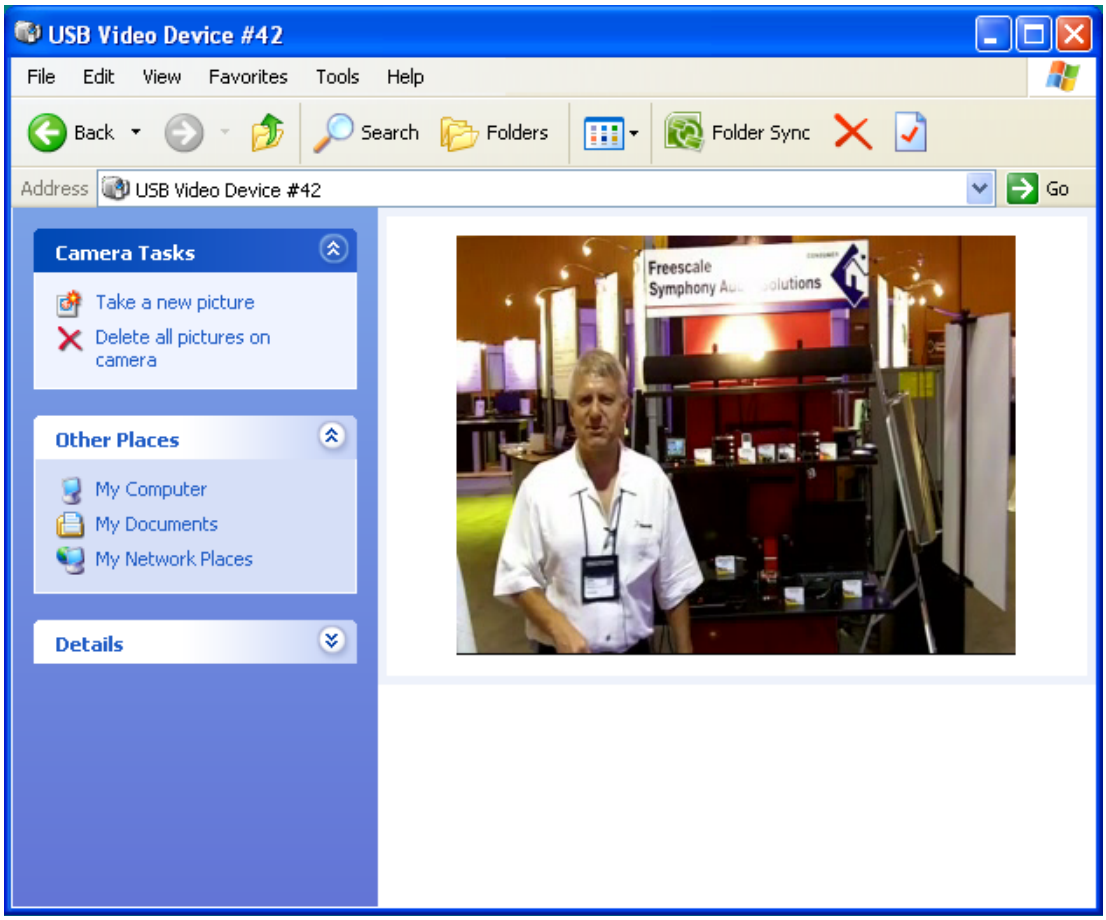


Figure J-15. Video is displayed

Appendix K MSD and CDC Composite Demo

K.1 Introduction

K.1.1 About MSD and CDC demo

This section gives a quick overview on how to use the USB Composite Device application: MSD_CDC composite device demo. In that, the MSD device contains the driver file which will be used to install CDC device.

By this way, user always have driver file of CDC device but does not need find it.

The USB MSD and CDC demo is developed base on USB MSD, CDC classes and consists of two different applications:

- USB MSD disk: receive request from host, send and receives data from host.
- USB CDC virtual com: receive request and data from host. The data, then can be displayed on HyperTerminal.

Because CDC feature include two interfaces, the application used IAD to write the feature's descriptor

To take you through this guide, the demo is illustrated by using a TWR K60N512 board.

K.2 Setting up the demo

Figure K-1 describes the demo setup. The TWR-K60N512 board is connected to a personal computer using two USB cables and one speaker through an external circuit. The computer uses one USB cable to supply power to the board and program the image to the flash. The computer also acts as the host system and the TWR-K60N512 board acts as the USB device. They are connected by the second cable.

The external circuit is connected to pin 40 on A side expansion port of dummy tower and is used to filter audio signal before entering the speaker.

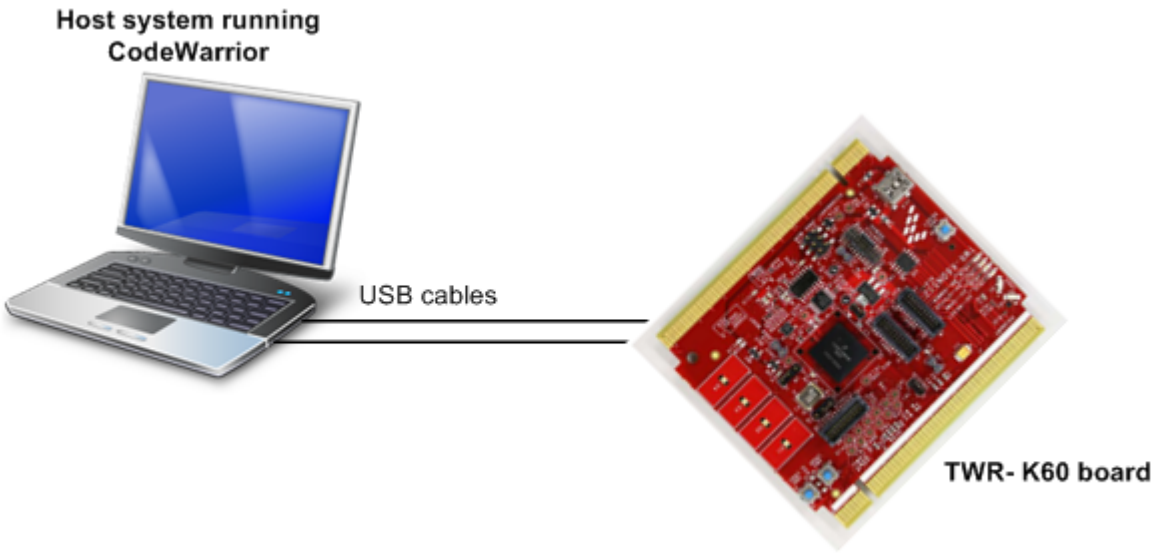


Figure K-1. MSD and CDC demo setup

K.3 Running the demo

After the system has been set, you must follow these steps to run the demo:

1. Plug USB Composite device in to the PC (host). As soon as you turn on the device, it is recognized by the host and is installed automatically. You must see the callout as shown in Figure K-2 on the right bottom corner of your screen.

Figure K-2. Find New Hardware Callout

Figure K-3. Find Composite device Callout

Then each feature device will be detected respectively:

The first, CDC device is detected:

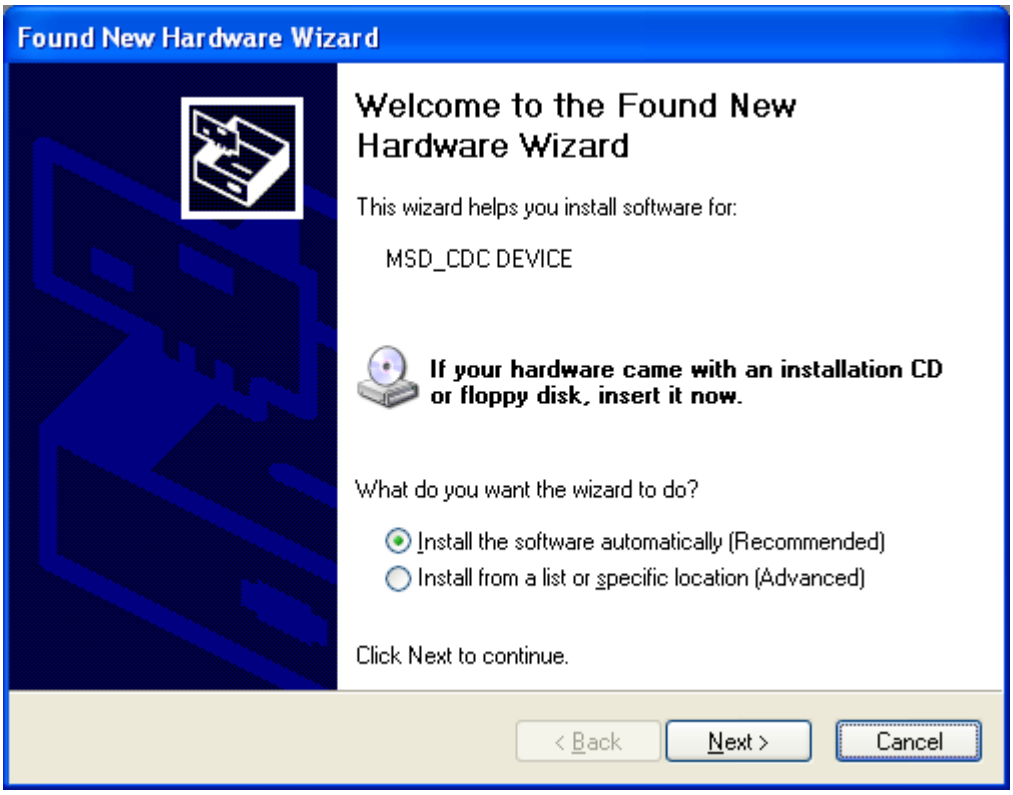


Figure K-4. Found new hardware Wizard

You should cancel to pass this request and install MSD feature device first

Next, Host detects MSD device.

Figure K-5. Find MSD device Callout

The host will install MSD feature device as a removable disk

Figure K-6. MSD feature device

2. The host will warning for CDC installation fail

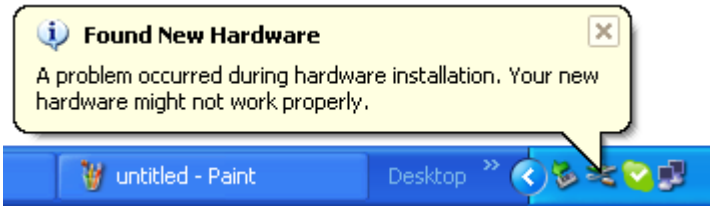


Figure K-7. hardware might not work properly

3. After that, MSD device is enumerated successfully. The FSL_MSDDEMO Removable disk can be seen at My Computer window as Figure K-8:

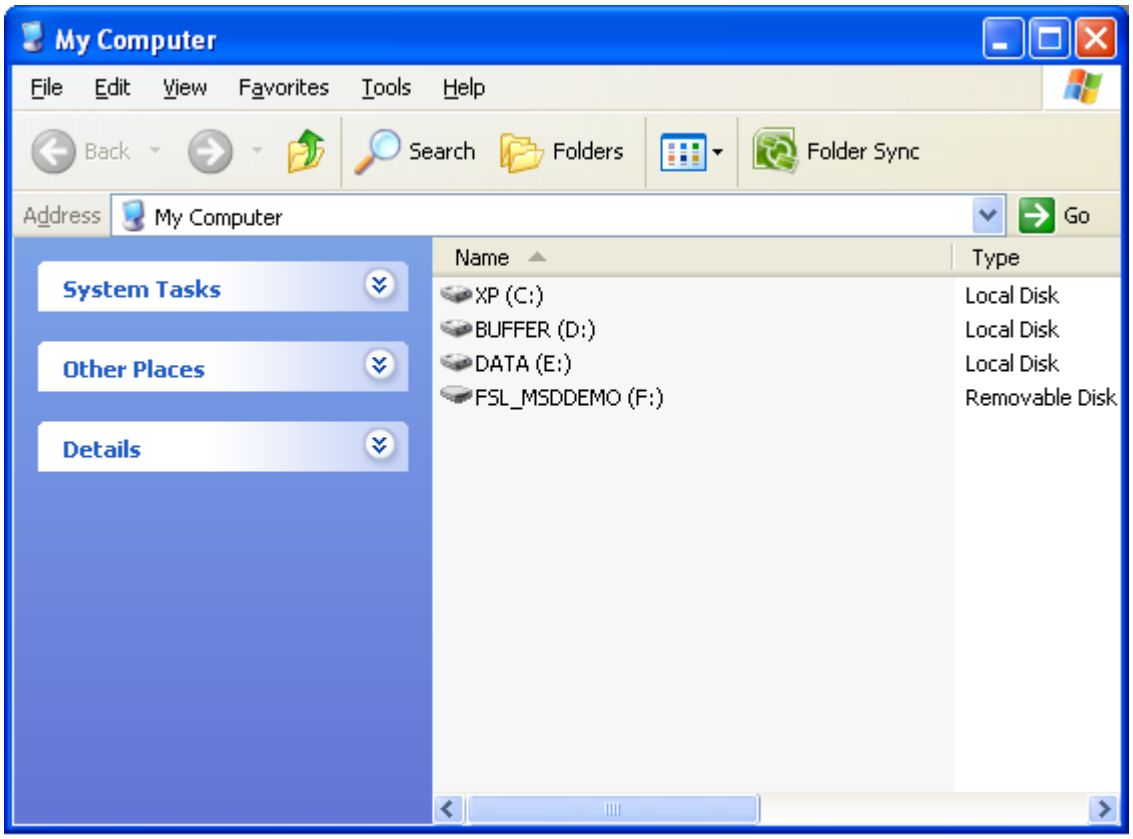


Figure K-8. FSL_MSDDEMO disk

This drive contains driver file for CDC, FSL_VCOM.INF which can be used to install for the CDC device:

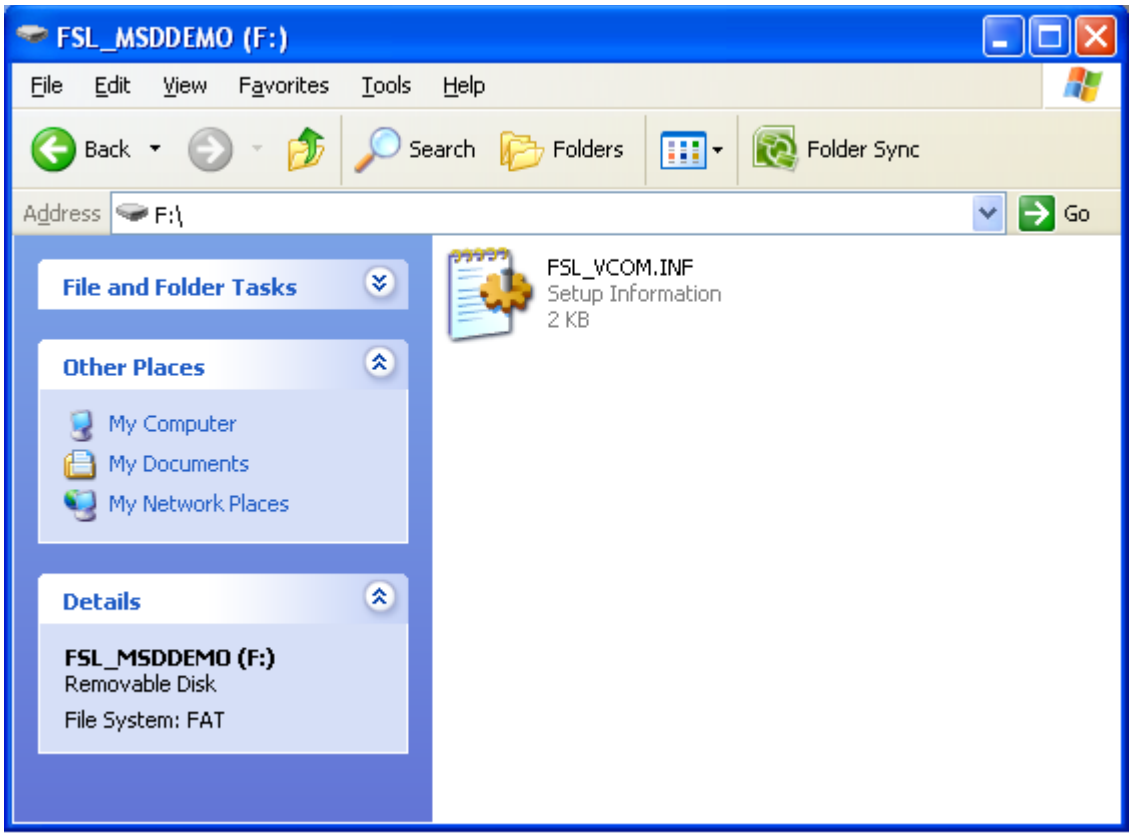


Figure K-9. CDC driver file

4. The steps to install CDC device is shown below:

Step 1. Update driver for CDC device

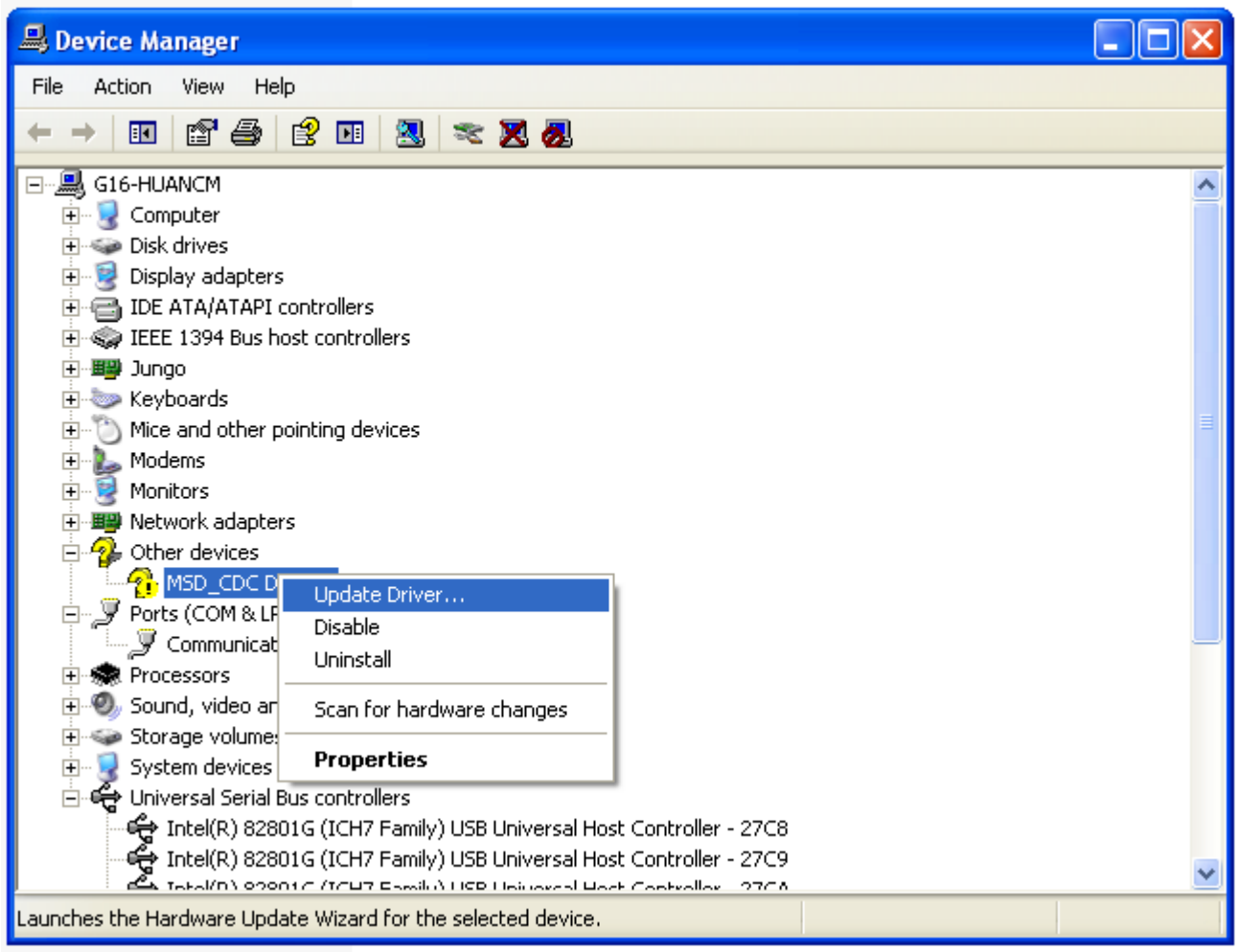


Figure K-10. Update Driver for CDC device

Step 2. Select Install from a list or specific location (Advanced) option and click on the Next button in Search and installation options window appears as shown in Figure K-11

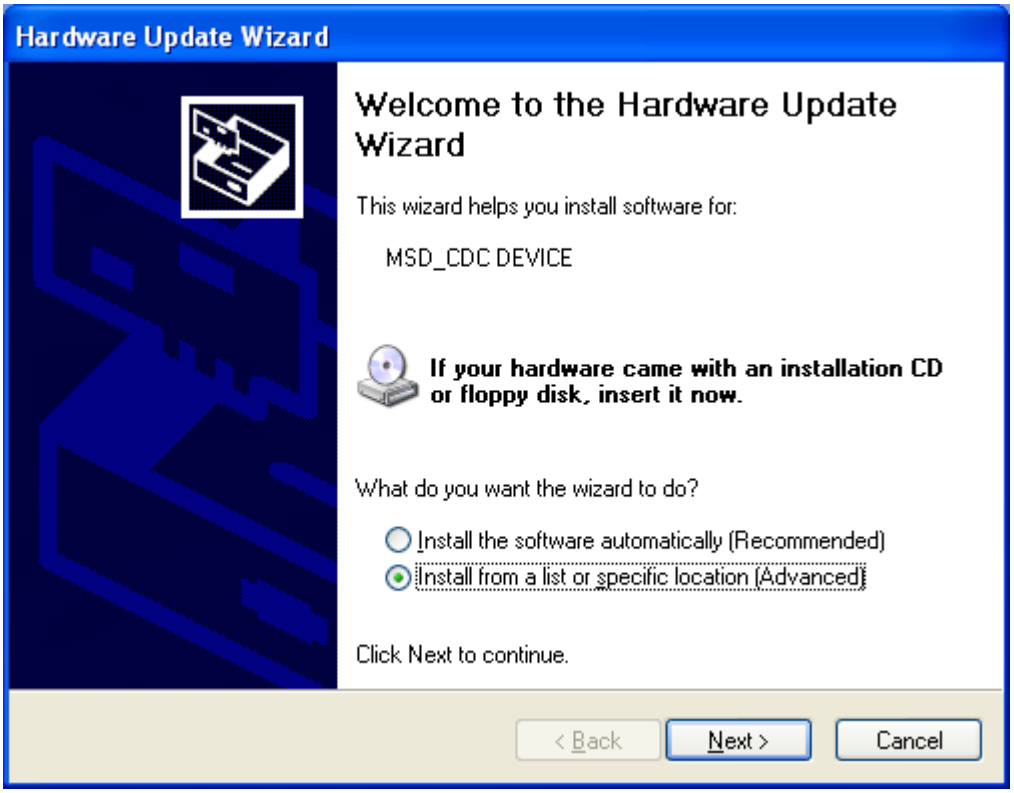


Figure K-11. Found new hardware

Step 3. Select "Don't search, I will choose the driver to install" option and click Next button as Figure K-12

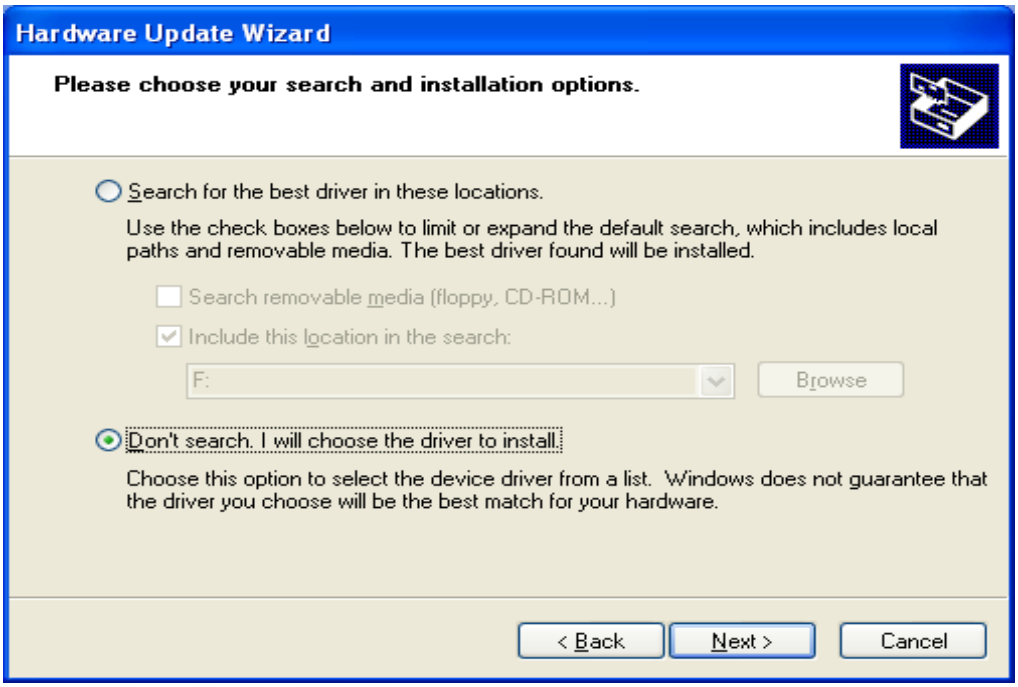


Figure K-12. Search and installation options

Step 4. Hardware Type Window appears, select Show All Devices option, and click Next button. Click Have Disk button when Select device driver window appears (Figure K-13).

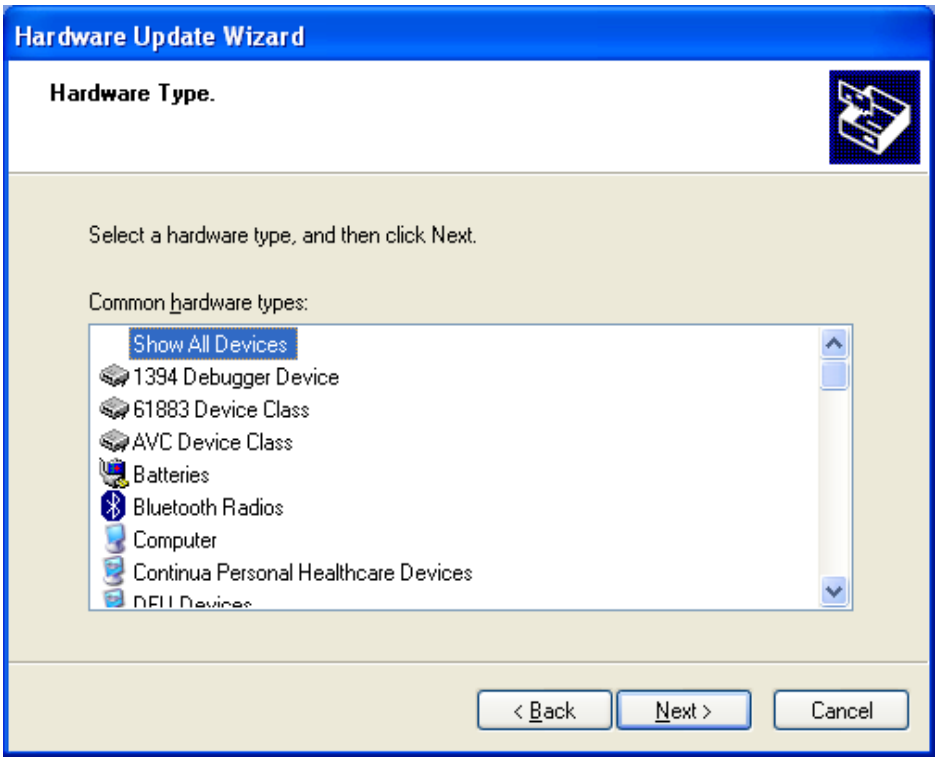


Figure K-13. Hardware type window

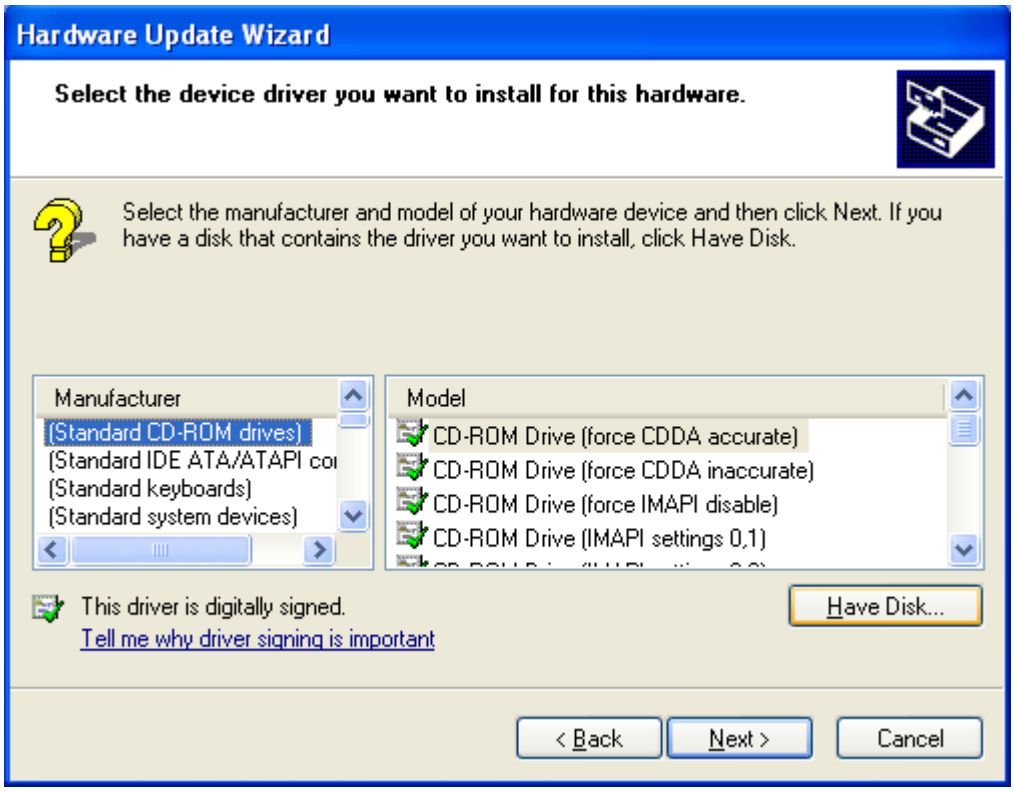


Figure K-14. Select device driver window

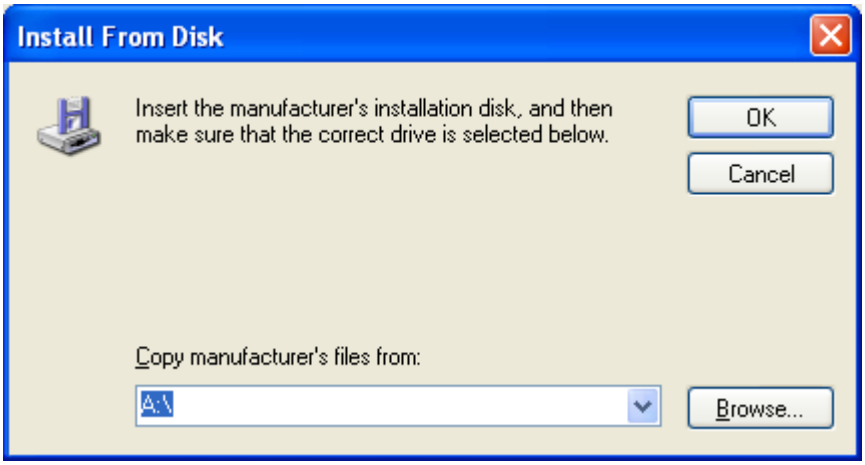


Figure K-15. Browse to driver

Step 5. Navigate to FSL_MSDDEMO disk, CDC driver file is FSL_VCOM.INF, choose it and click Open button (Figure K-16). After that, click Next button when the next window appears to install driver.

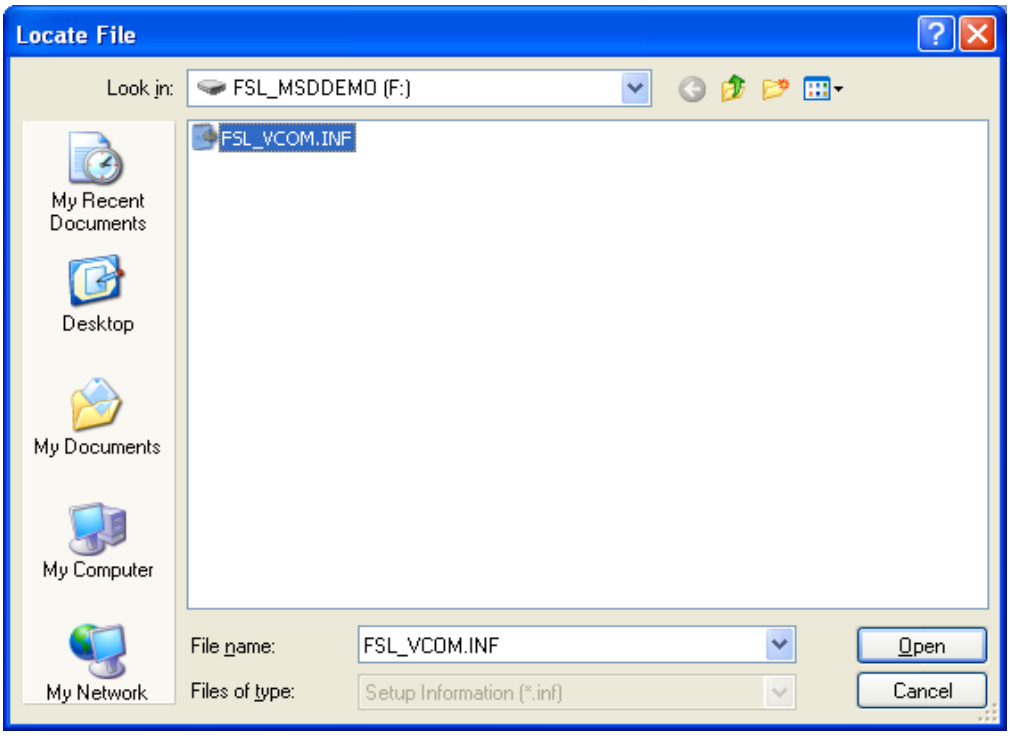


Figure K-16. Location to the driver

Step 6. The device in this application is not supported yet by Window. Click Yes button in Update driver warning window (Figure K-17)and Continue Anyway button in next window(Figure K-18).

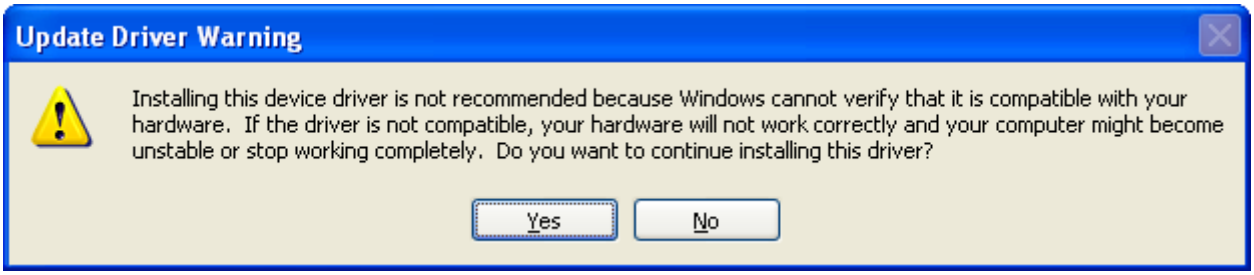


Figure K-17. Update Warning

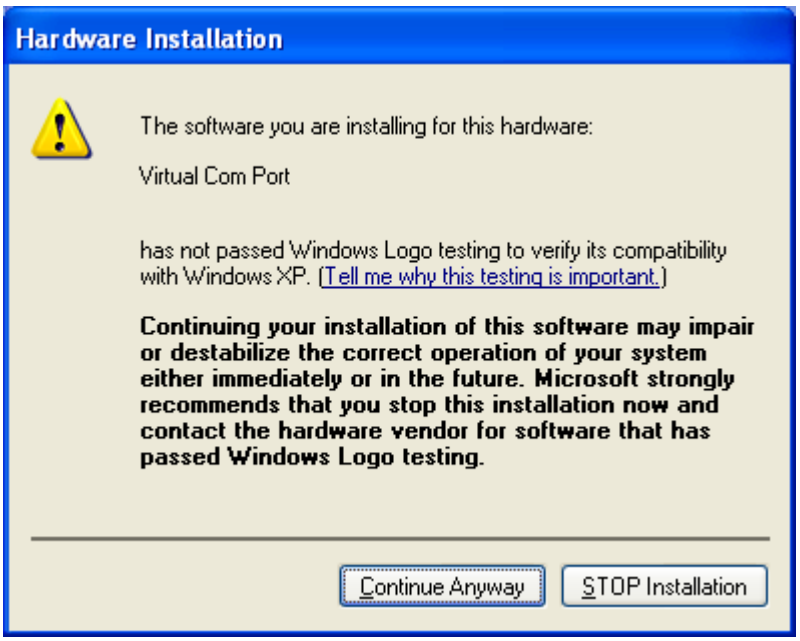


Figure K-18. Hardware Installation Confirm

Step 7. Click to Finish button when installation for CDC device done

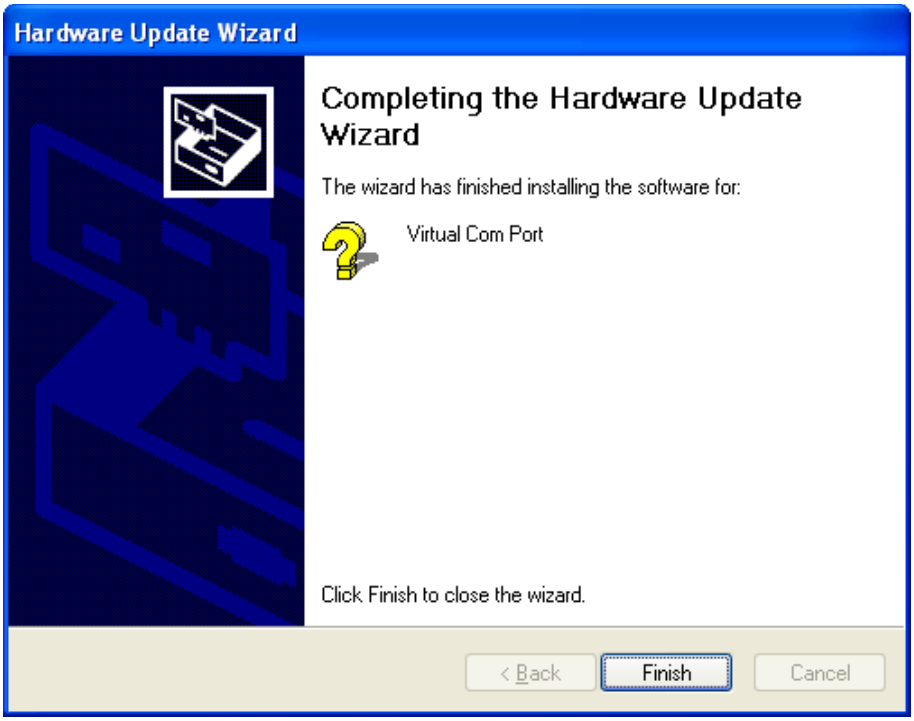


Figure K-19. Update complete

Step 8. After successful installation, the host indicates that the device is ready to use as shown in Figure K-20.

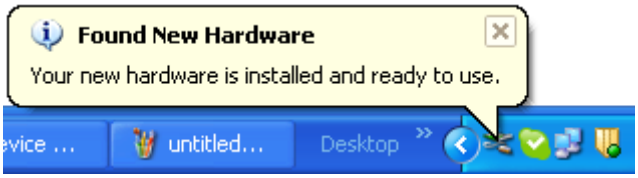


Figure K-20. Installation of USB Composite device

5. To verify whether the USB Composite Device has been installed properly or not, you must see the each feature devices and composite device entry in the device manager as shown in Figure K-21.

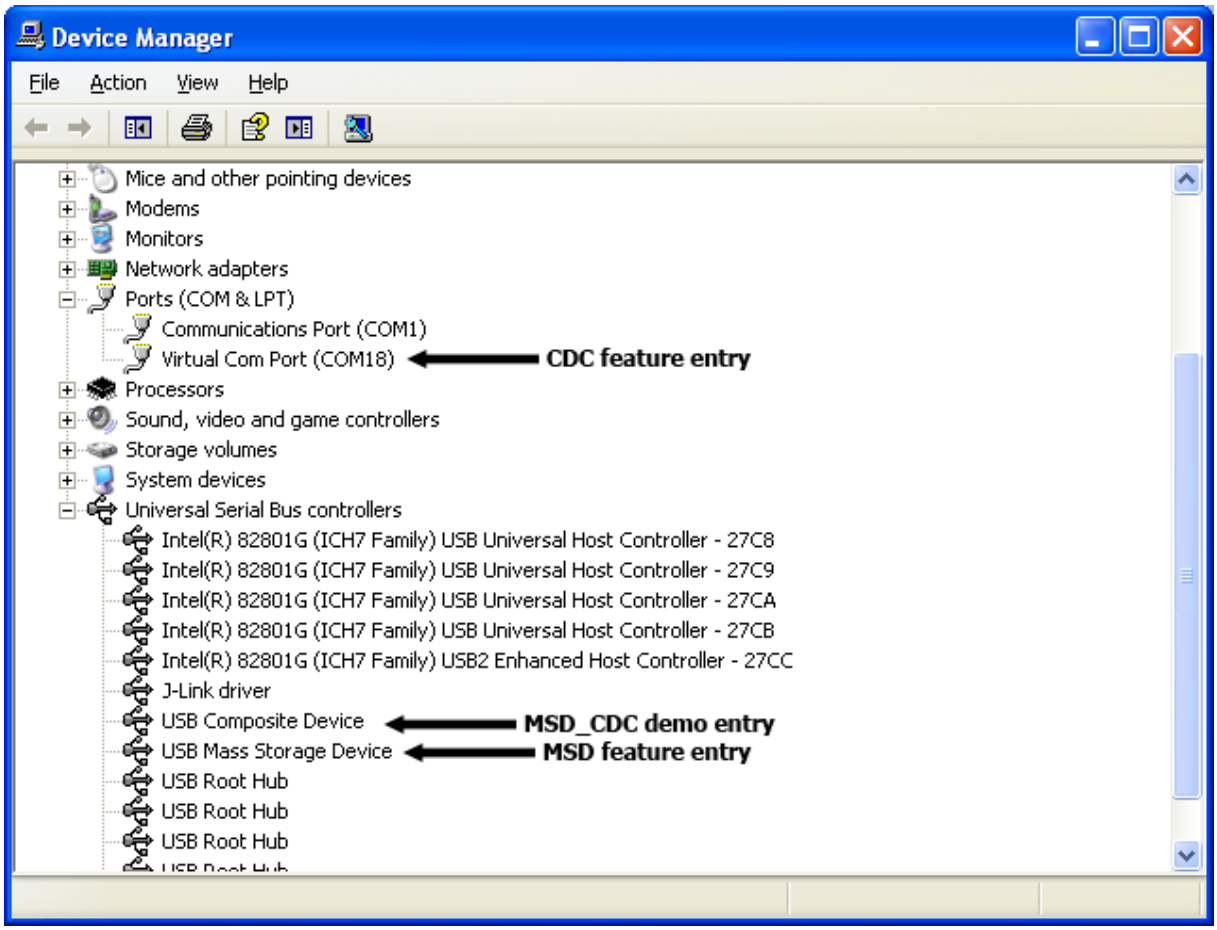


Figure K-21. Device manager dialog

6. Double-click on the Virtual Com Port icon, Virtual Com Port Properties dialog appears as shown in Figure K-22.

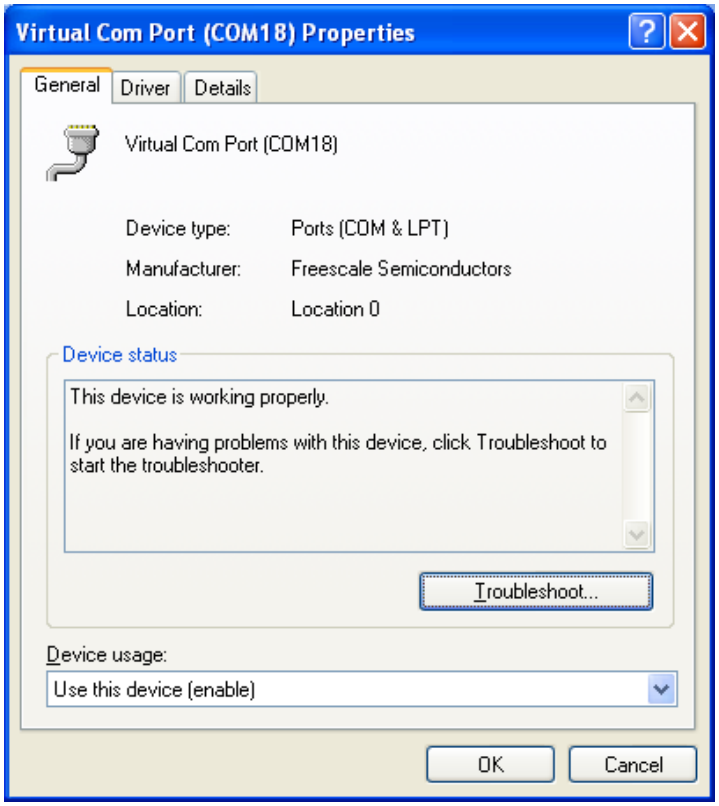


Figure K-22. USB Video Device Properties dialog

7. Configure HyperTerminal for MSD_CDC_DEMOE com port step by step, following:

Step 1.Open Run task and enter hypertrm.exe as shown in Figure K-23.

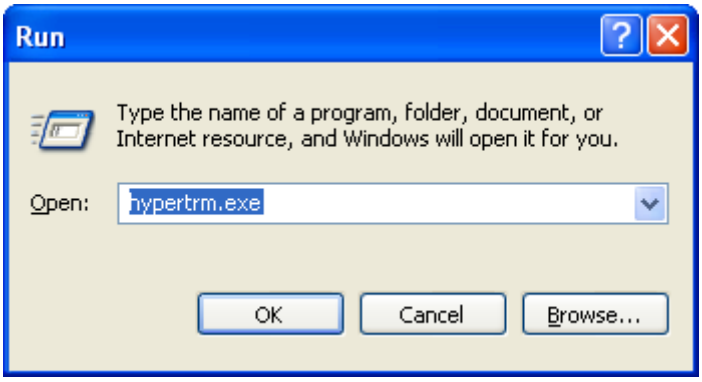


Figure K-23. Open HyperTerminal from Run task

Step 2.Enter name of connection as Figure K-24.

Figure K-24. HyperTerminal startup

Step 3.Then choose port to connect. In this application, the com is COM18 (Figure K-25).

Figure K-25. Choose Com port

Step 4. After selecting the COM port, configure baud rate and other properties as shown in Figure K-26 (Baud rate: 115200, Data bits: 8, Parity: None, Stop bits: 1, Flow control: None)

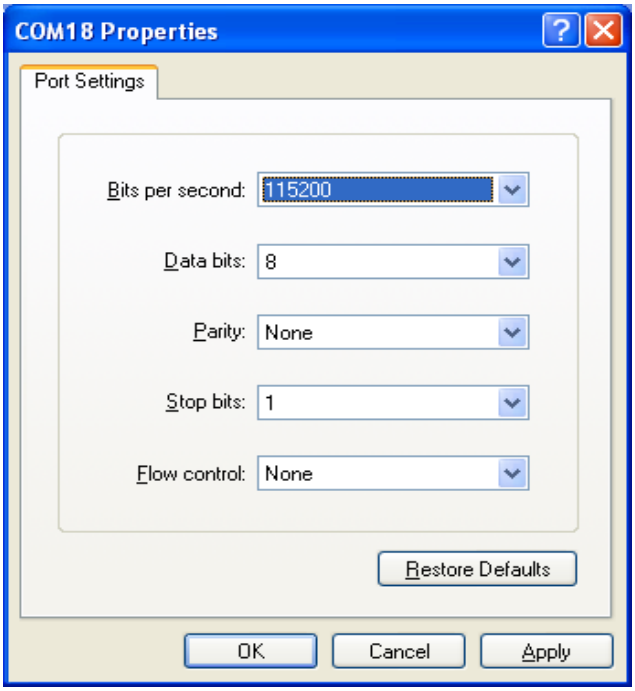


Figure K-26. COM properties

Step 5.Configure this port as shown as in Figure K-27. Enter to ASCII setup and setting as shown as in Figure K-29

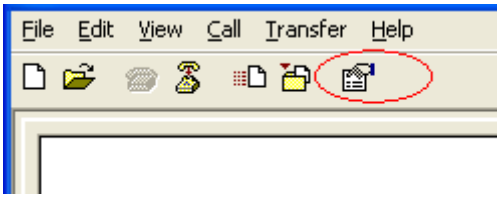


Figure K-27. Port taskbar

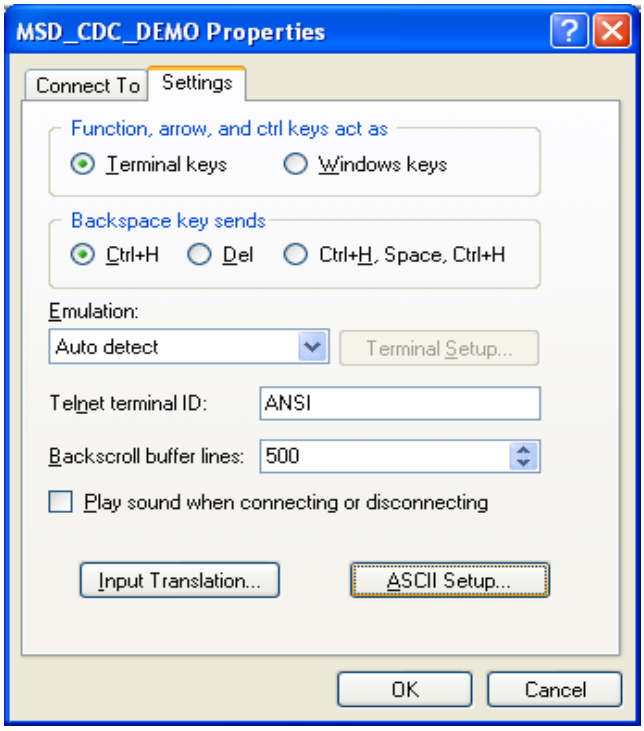


Figure K-28. Com port properties

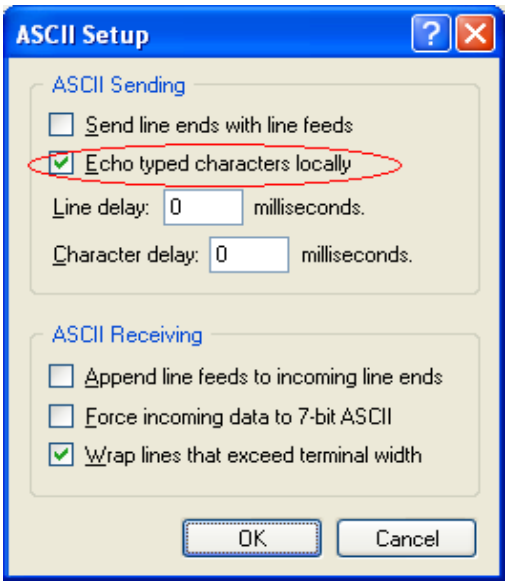


Figure K-29. Setting Echo character

Step 6. The HyperTerminal is configured now as shown in Figure K-30

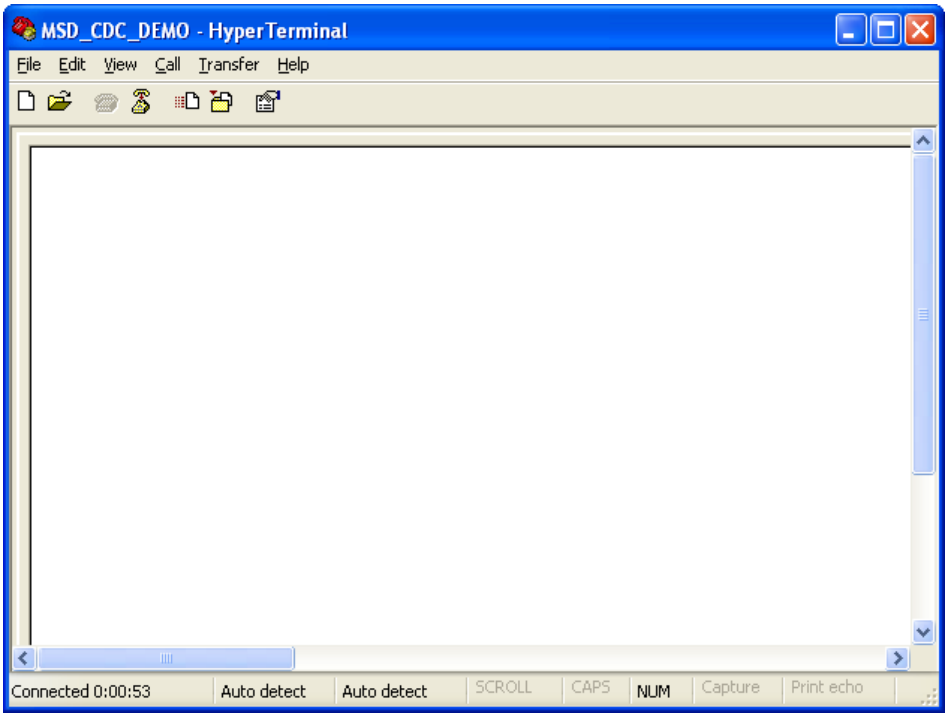


Figure K-30. HyperTerminal

8. In MSD_CDC_DEMO window enter: "MSD_CDC_DEMO " string. The string will display on HyperTerminal window as shown as in Figure K-31

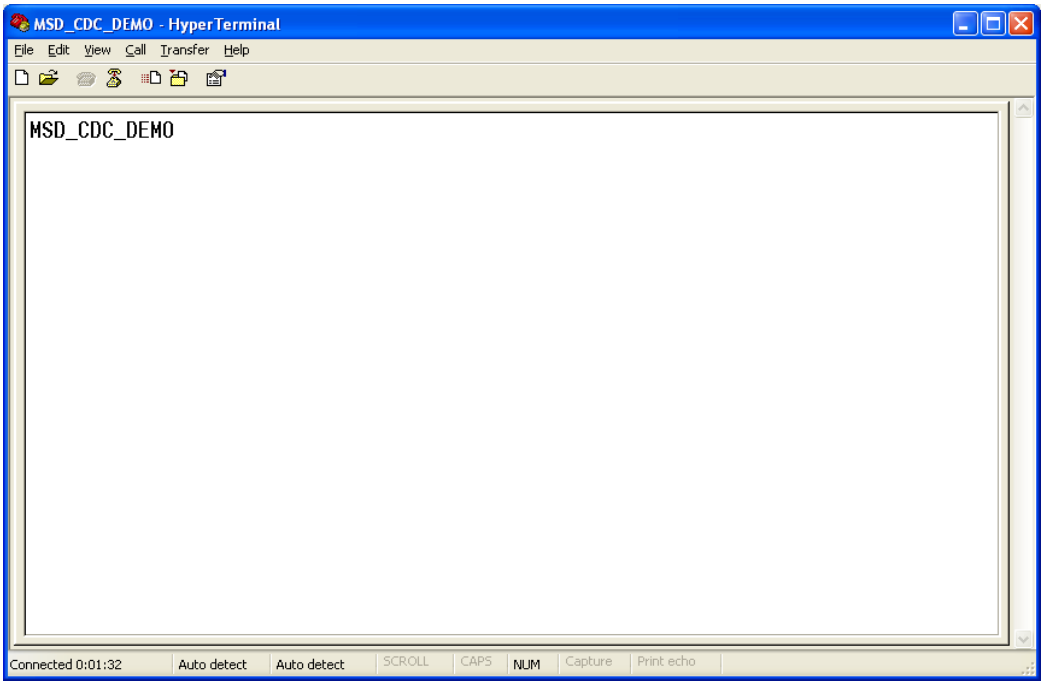


Figure K-31. MSD_CDC_DEMO HyperTerminal

Appendix L HID Audio Video Composite Demo

L.1 Introduction

L.1.1 About HID_Audio_Video demo

This section gives a quick overview on how to use the USB Composite layer application USB HID, Audio, Video classes device demo software package.

The USB HID_Audio_Video demo is developed base on USB HID, Audio, Video classes and consists of three different applications:

- USB HID mouse: receive request from host and send mouse data to the host.
- USB Audio speaker: receive request and audio stream data from host. The audio data then can be play on the device.
- USB Video virtual camera: receive request from host and send video stream data to the host.

Because both video and audio feature include two interfaces, the application used IAD to write these feature's descriptor

To take you through this guide, the demo is illustrated by using a TWR K60 board.

L.2 Setting up the demo

Figure L-1 describes the demo setup. The TWR-K60N512 board is connected to a personal computer using two USB cables and one speaker through an external circuit. The computer uses one USB cable to supply power to the board and program the image to the flash. The computer also acts as the host system and the TWR-K60N512 board acts as the USB device. They are connected by the second cable.

The external circuit is connected to pin 40 on A side expansion port of dummy tower and is used to filter audio signal before entering the speaker.

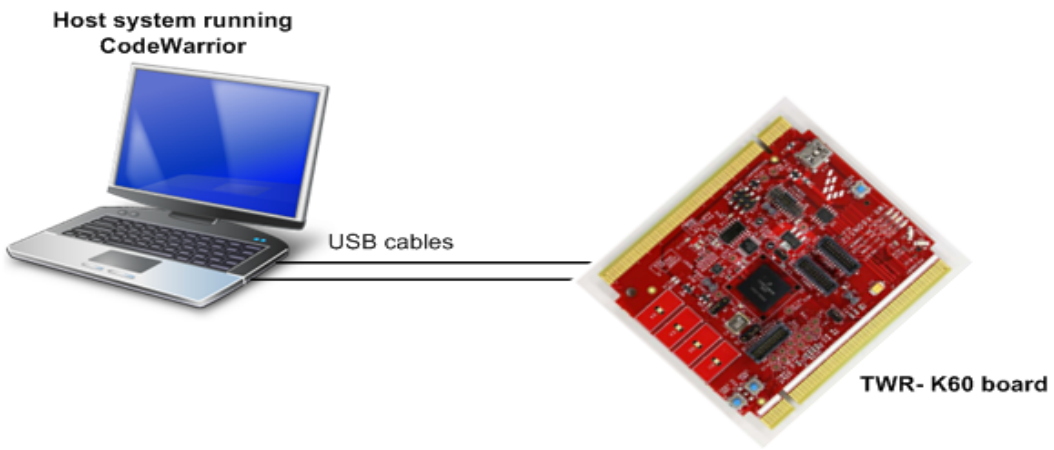


Figure L-1. HID Audio Video demo setup

L.3 Running the demo

After the system has been set, you must follow these steps to run the demo:

1. Plug USB Composite device in to the PC (host). As soon as you turn on the device, it is recognized by the host and is installed automatically. You must see the callout as shown in Figure L-2 on the right bottom corner of your screen.

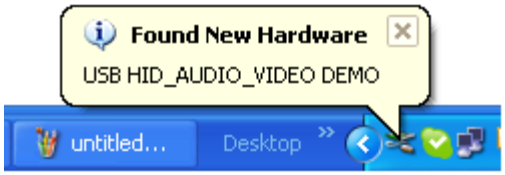


Figure L-2. Find New Hardware Callout

Figure L-3. Find Composite device Callout

Then each feature device will be detected respectively:

Firstly, Video device is detected:

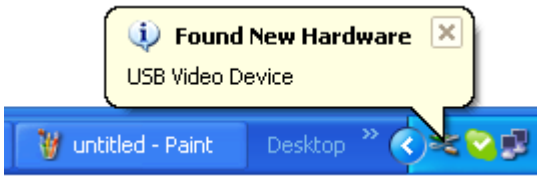


Figure L-4. Find Video device Callout

Next, Audio device will be detected

Figure L-5. Find Audio device Callout

HID device will be detected lastly

Figure L-6. Find HID device Callout

2. After successful installation, the host indicates that the device is ready to use as shown in Figure L-7

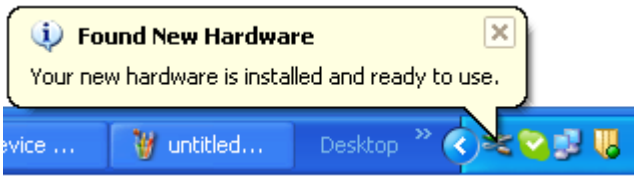


Figure L-7. Installation of USB Composite device

3. To verify whether the USB Composite Device has been installed properly or not, you must see the each feature devices and composite device entry in the device manager as shown in Figure L-8.

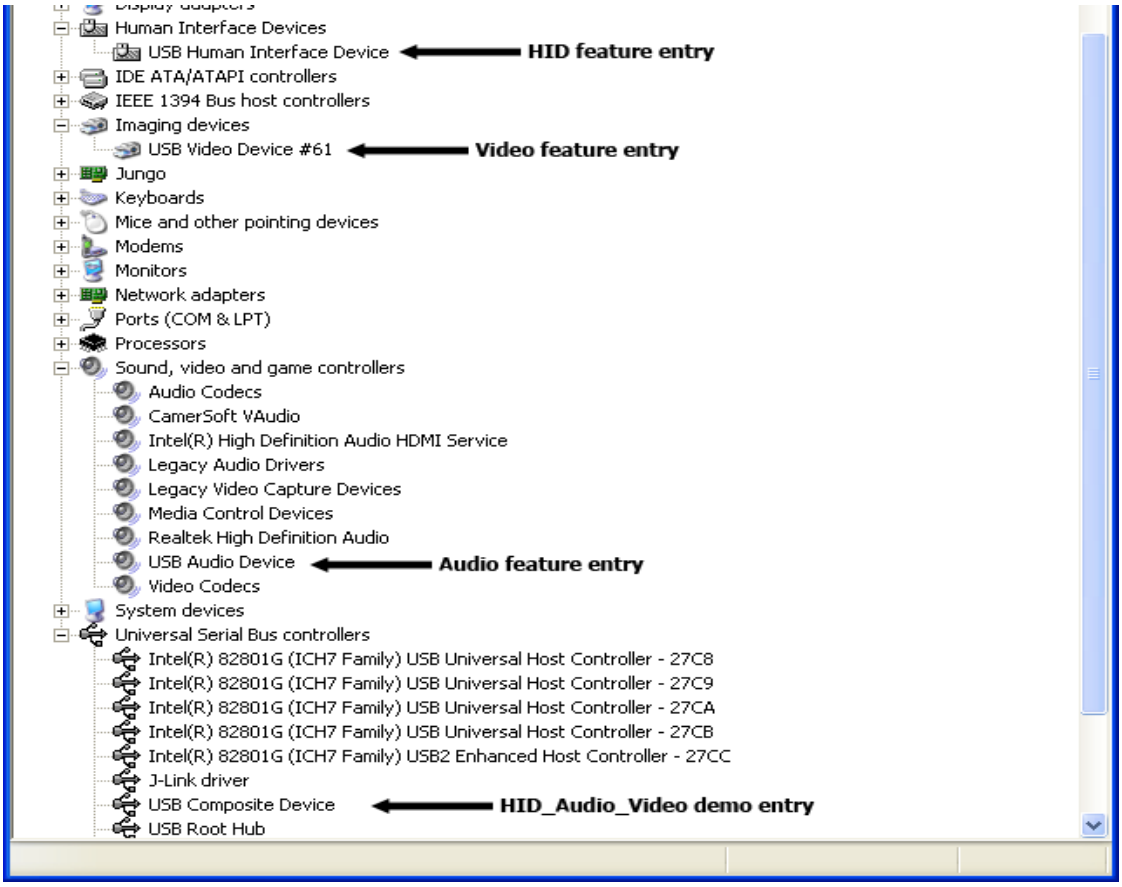


Figure L-8. Device Manager

L.4 Video virtual camera feature demo

1. Double-click on the USB Video device icon, USB Video device Properties dialog appears as shown in Figure L-9.

Figure L-9. USB Video Device Properties dialog

2. To verify whether the USB Video Device has been selected as the default device or not, you open My Computer icon. The Virtual camera will be identified as a Digital camera and appears as shown in Figure L-10

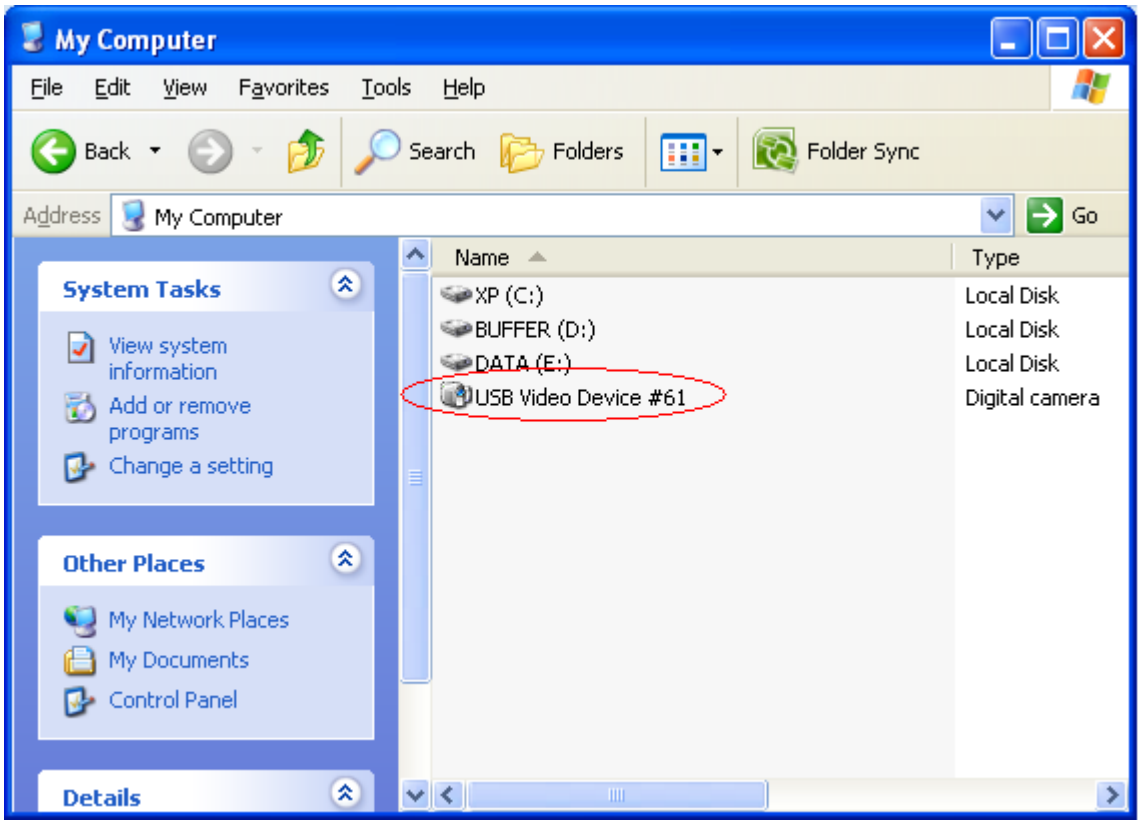


Figure L-10. Digital camera application

3. Double-click on Digital camera icon video is displayed as shown in Figure L-11.

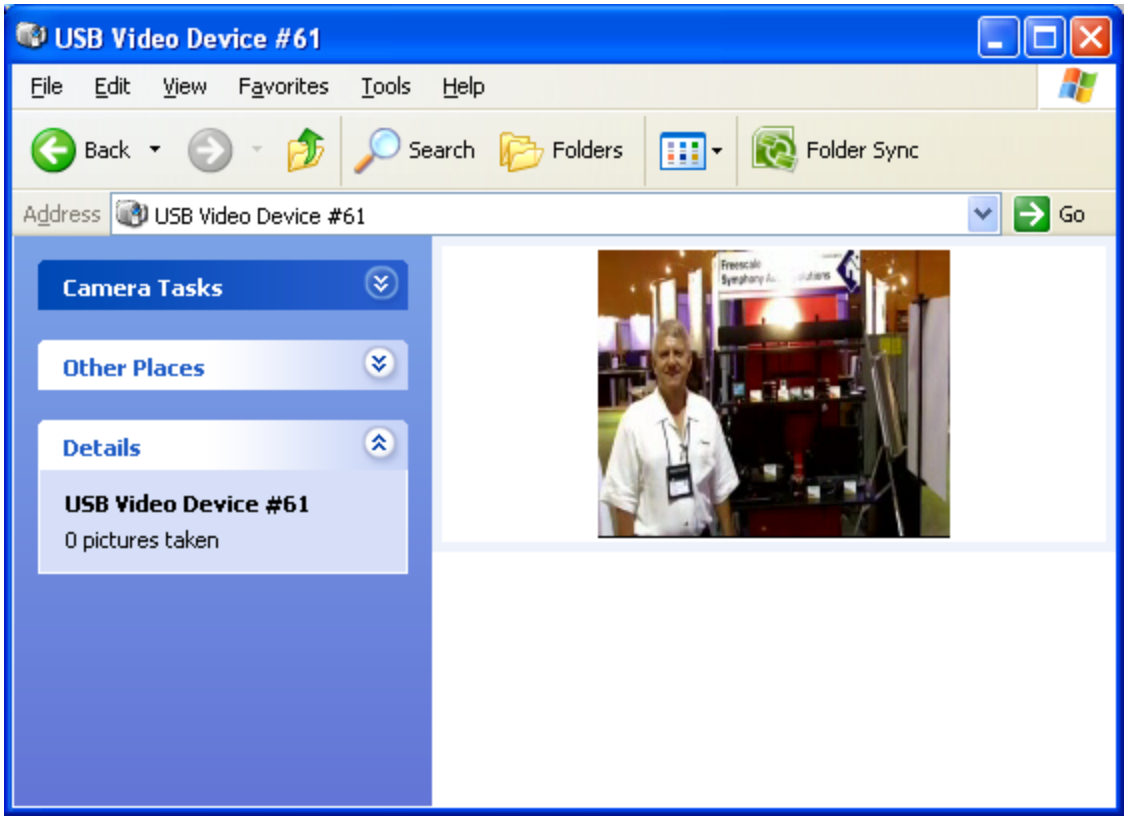


Figure L-11. Virtual camera

L.5 Audio speaker feature demo

1. Double-click on the USB Audio device icon, USB Audio device Properties dialog appears as shown in Figure L-12

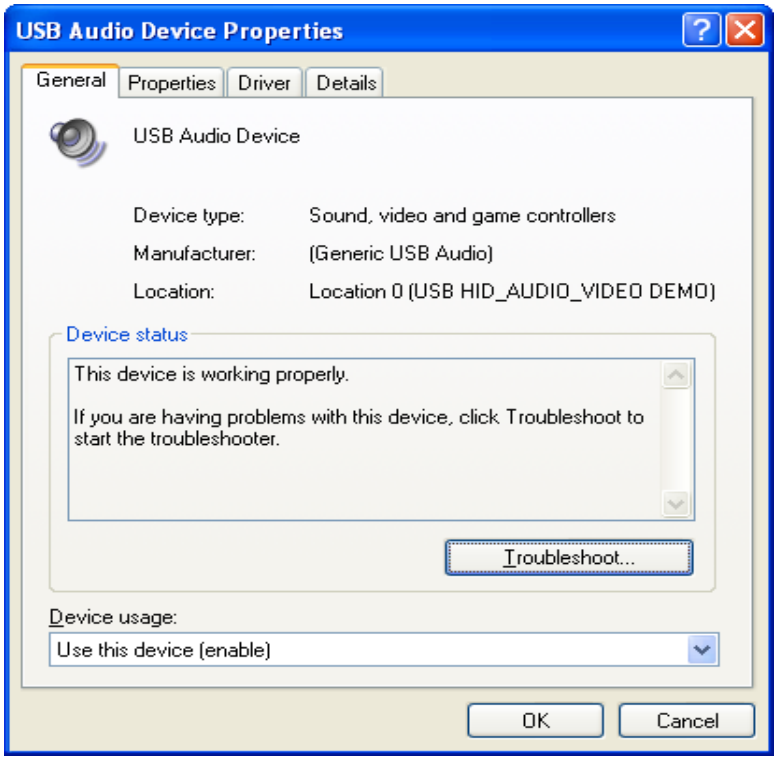


Figure L-12. USB Audio Device Properties dialog

2. To verify whether the USB Audio Device has been selected as the default device or not, you must right-click on master volume icon. Master volume dialog appears as shown in Figure L-13

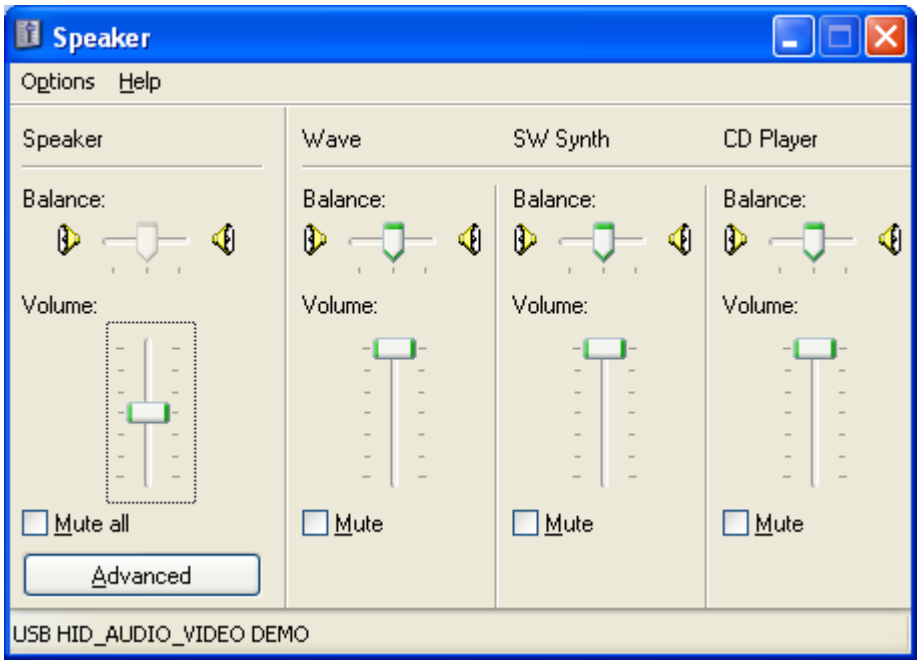


Figure L-13. Master volume

3. Open the Window Media Player application as shown in Figure L-14.

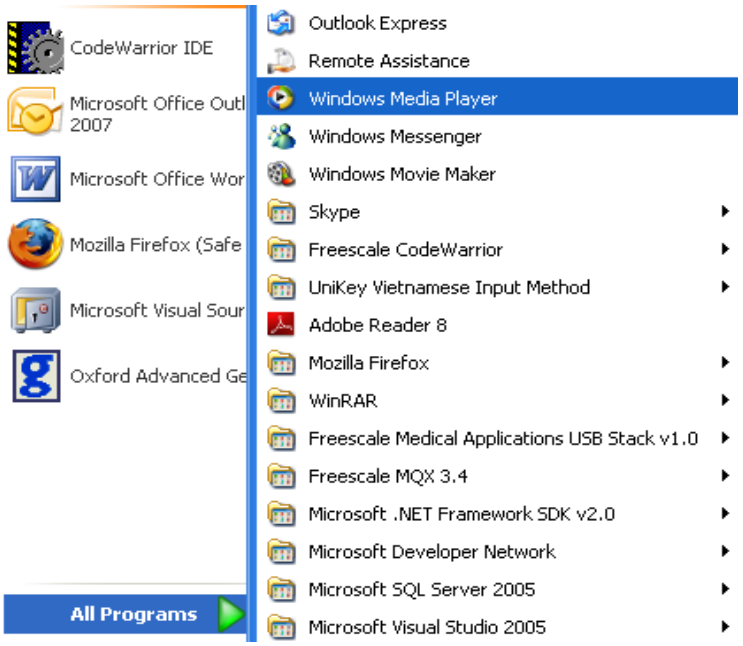


Figure L-14. Launch Windows Media Player application

4. The window shown in the Figure L-15 appears. Select and play your favorite song, you can hear the song clearly.

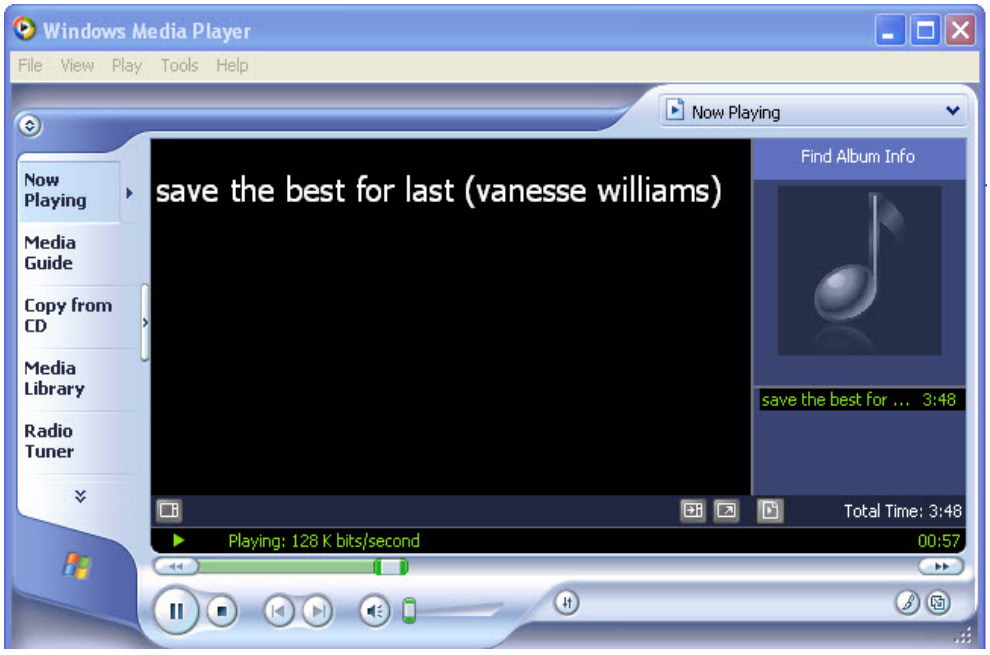


Figure L-15. Windows Media Player

L.6 HID mouse feature demo

1. To verify whether the mouse has been properly installed or not, you must see the Mouse demo device entry in the device manager

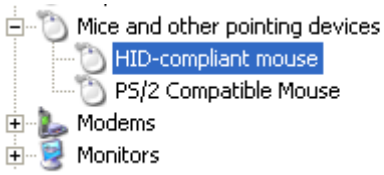


Figure L-16. Mouse demo device entry

2. After the HID device is installed, it can be moved on the host computer screen by pressing the push buttons. Figure L-17 shows the function of these button.

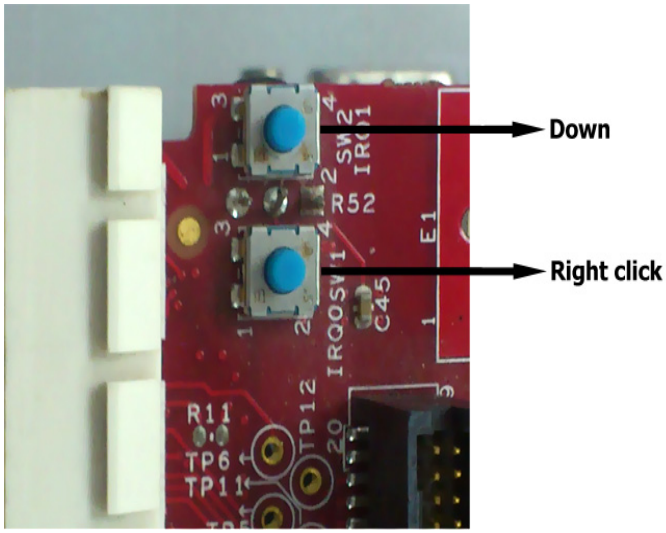


Figure L-17. HID device push buttons

Glossary

A

API

Application Programming Interface. It is a set of functions, structures, and classes that can be called from an application program to access other programs. In S08USB applications, it is an interface defined by a layer that the application developer can use.

application
A user program that you can develop using the driver layers to performs a specific function.

C

callback
An application-provided function that S08 USB software calls, when a particular event occurs.

CDC
Communication Device Class. This device class is used for implementation of communication protocols over the USB transport.

class driver
A driver that can operate large number of devices of the comparable type.

CodeWarrior
An integrated development environment used to develop software.

configuration descriptor
The structure that defines the configurations for a USB device. A USB device can have one or more configuration descriptors. Each configuration has one or more interfaces and each interface has zero or more endpoints. An endpoint is not shared among different interfaces within a single configuration, however it can be shared among interfaces that are part of different configurations without this restriction.

Continua Alliance
The alliance formed between various companies for the development of medical connectivity standards.

control endpoint
The data pipe defined in the USB protocol that is used to receive and transmit control data.

controller
The hardware module that controls the bus to transmit or receive data.

D

debugger
The combination of hardware and software used in the development of a project to find and resolve issues.

descriptor
A data structure that contains the information about the USB device and its features.

device descriptor
The structure that describes a USB device. It includes configurations and the information that apply to a USB device. A USB device has only one device descriptor.

device driver
It is the software that controls the controller device and drives it to transmit or receive data.

device stack
These are the group of software layers through which the data passes to the application.

E

enumeration
This is a process in USB protocol by which the host identifies the devices connected to it.

event
It is a condition occurring at the bus that is passed through the device stack to the upper layers of software.

endpoint
It signifies logical data source and sink of a USB device.

endpoint designator
It contains details required by the host to determine the bandwidth requirements of endpoints.

F

function parameters
The parameters passed on a function call.

H

HID
Human Interface Device. This is the devices that is used by humans to control the operation of computer systems. Typical examples include keyboard, joystick, and mouse.

host
The hardware and its operating system (for example, a desktop personal computer), where you develop your application.

I

IEEE11073
It is an IEEE standard that defines medical connectivity standards between devices and applications at various levels.

I/O
Generally refers to the transfer of commands or data across a device interface.

M

MSC
Mass Storage Class. The USB mass storage device class (USB MSC or UMS) is a set of computing communications protocols defined by the USB Implementers Forum that run on the USB. The standard provides an interface to a variety of storage devices like USB flash drive, memory card reader, digital audio player, digital camera, external drive and so on.

P

PHD
Personal Healthcare Device. This is the devices that is used to take medical measurements. Typical examples include pulse meter, glucometer, and blood pressure monitor.

porting
It is a process of moving some applications and software layers that work on one device platform to another.

R

resume
The process of waking a USB device from suspend state so that it can start sending data.

S

setup packet
A special 8 byte packet that the host sends on the control endpoint to receive control information from the device.

suspend
A phase in the USB protocol where the device and the host turn off their transmitters to save power if there is nothing to transmit on the USB bus.

SCSI
Small Computer System Interface. It is a set of standards for physically connecting and transferring data between computers and peripheral devices. SCSI is generally pronounced as *scuzzy*.

T

transport
Moving data from one entity to another.

U

USB
Universal Serial Bus. It is a serial bus that is used to connect devices to a personal computer.