Freescale USB Host Stack

Users Guide

Document Number: USBHOSTUG
Rev. 6
03/2012

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

freescale™
semiconductor

Document Number: USBHOSTUG
Rev. 6
03/2012

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	04/2010	Launch release.
Rev. 2	06/2010	• Added support for CFV2 devices. • Rebranded Medical Applications USB Stack Host to Freescale USB Stack with PHDC Host.
Rev. 3	09/2010	• Added support for CodeWarrior 10 • Fig 2-1: Freescale USB Stack with PHDC — Host Directory Structure updated
Rev. 4	01/2011	• Added Audio Host demo
Rev. 5	07/2011	• Updated document name to USBHOSTUG. • Updated figures in Appendix A
Rev. 6	03/2012	• Added 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

Appendix E
Audio Host Demo

Appendix F
USB FAT File System Demo

Chapter 1 Before You Begin

1.1 About Freescale USB Stack — Host Architecture

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 microphones, keyboards, storage devices, cameras, printers, and many others. USB interconnects are also becoming more and more popular in the medical segments.

1.2 About this book

This book describes the Freescale USB Stack — Host Architecture class functionality. The following table shows the summary of chapters included in this book.

Table 1-1. USBHOSTUG summary

Chapter Title	Description
Before You Begin	This chapter provides the prerequisites for reading this book.
Getting Familiar	This chapter provides the information about the Freescale USB stack with PHDC software suite.
Freescale USB stack with PHDC — Host Architecture	This chapter discusses the architecture design of the Freescale USB suite.
Developing applications	This chapter provides the steps a developer must take to develop applications on top of the pre-developed classes.
Working with the software	This chapter provides the steps to building, running the applications.
Human Interface Device (HID) Demo	This chapter provides the setup and running HID demo for CFV1 and CFV2 processors.
Virtual Communication (COM) Demo	This chapter provides the setup and running CDC demo for CFV1 and CFV2 processors.
Mass Storage Device (MSD) Demo	This chapter provides the setup and running MSD demo for CFV1 and CFV2 processors.
USB Audio Host Demo	This chapter provides the setup and running the USB Audio Host demo

1.3 Reference material

Use this book in conjunction with:

- *Freescale USB Stack API Reference Manual* (document USBAPIRM)
- USB Host source code
- ColdFire V2 USB Device Source Code

We assume that you are familiar with the following reference material:

- USB Specification Revision 1.1
- USB Specification Revision 2.0
- USB Device Class Definition for Audio Devices Revision 1.1
- MCF52259 Reference Manual
- CodeWarrior Help

1.4 Acronyms and abbreviations

Table 1-2. Acronyms and abbreviations

API	Application Programming Interface
CDC	Communication Device Class
CFV1	ColdFire V1 (MCF51JM128 CFV1 device is used in this document)
CFV2	ColdFire V2 (MCF52221 and MCF52259 CFV2 devices are used in this document)
COM	Communication
HID	Human interface device
IDE	Integrated development environment
KHCI	Host Controller Interface
MSD	Mass storage device
PC	Personal computer
TR	Transfer Request
USB	Universal Serial Bus

1.5 Important terms

The following table shows the terms used throughout the book.

Table 1-3. Important terms

Term	Description
Class Driver	These offer a generic control interface for a family of devices.
Enumerate	A process in the USB protocol by which the host identifies the devices connected to it.
USB Low Level Drivers	USB low level drivers are the driver software layers that interface with hardware and abstract them for the class drivers.
USB Chapter 9 Request	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.

Chapter 2 Getting Familiar

2.1 Introduction

The Freescale USB Stack — Host Architecture contains the low level driver code, commonly used class drivers, and some basic applications. This document intends to help you gain an insight into the stack and capabilities to develop your own classes and applications. It is targeted for firmware application developers who would like to develop the applications using USB as the transport.

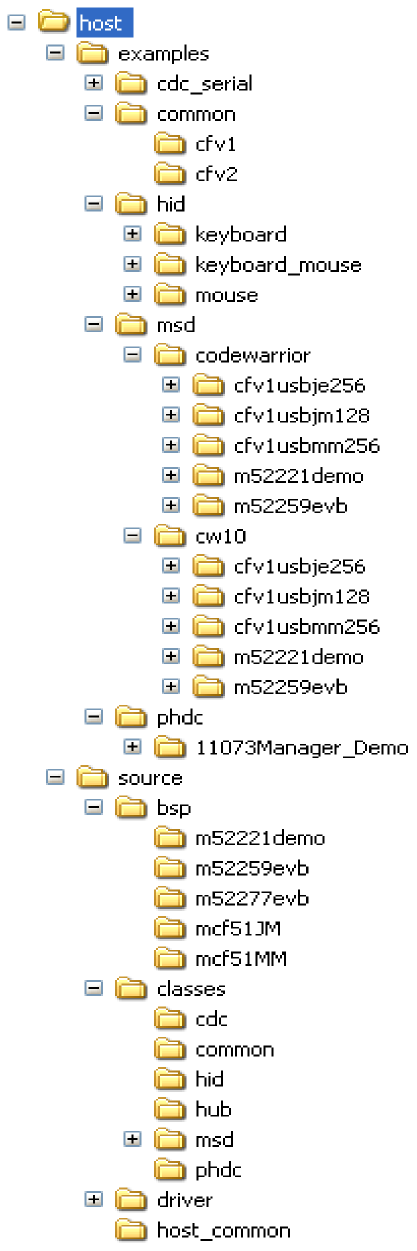
2.2 Software suite

The software suite comprises the USB low level drivers for the 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 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.

The following figure shows the directory structure:



```
host
  examples
    cdc_serial
    common
      cfv1
      cfv2
    hid
      keyboard
      keyboard_mouse
      mouse
    msd
      codewarrior
        cfv1usbje256
        cfv1usbjm128
        cfv1usbmm256
        m52221demo
        m52259evb
      cw10
        cfv1usbje256
        cfv1usbjm128
        cfv1usbmm256
        m52221demo
        m52259evb
    phdc
      11073Manager_Demo
  source
    bsp
      m52221demo
      m52259evb
      m52277evb
      mcf51JM
      mcf51MM
    classes
      cdc
      common
      hid
      hub
      msd
      phdc
    driver
    host_common
```

Figure 2-1. Freescale USB Stack — Host Directory Structure

Chapter 3 Freescale USB Stack — Host Architecture

3.1 Architecture overview

The purpose of the Freescale USB Stack — Host Architecture is to provide an abstraction of the USB hardware controller core. A software application written using the USB host API can run on full speed or low speed core with no information about the hardware. In the USB, the host interacts with the device using logical pipes. After the device is enumerated, a software application needs the capability to open and close the pipes.

After a pipe is opened with the device, the software applications can queue transfer in either direction and is notified with the transfer result through an error status. In short, the communication between host and device is done using logical pipes that are represented in the stack as pipe handles. Figure 3-1 shows each of the blocks as part of the host API.

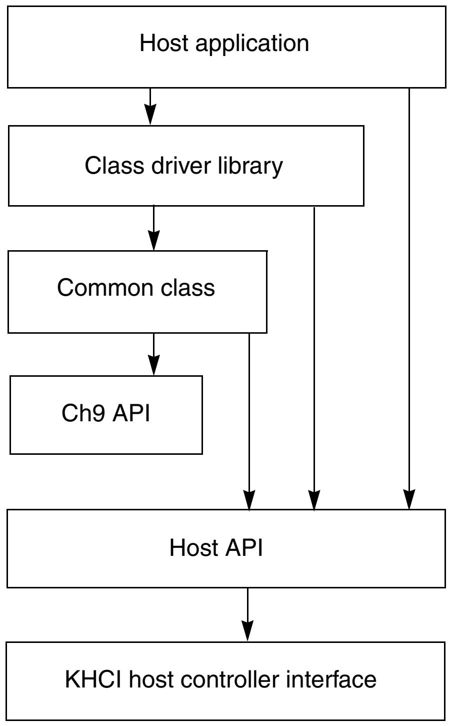


Figure 3-1. USB Host Architecture

3.2 Host application

A host embedded application software (also called a device driver) is implemented for a target device or a class of devices. The Freescale USB Host Stack comes with examples for a few classes of devices.

Application software can use API routines to start and stop communication with a USB bus. It can suspend the bus or wait for a resume signal for the bus.

3.3 Class-driver library

The class-driver library is a set of wrapper routines that can be linked into the application. These routines implement standard functionality of the class of device, defined by USB class specifications. It is important to note that even though a class library can help in reducing some implementation effort at the application level, some USB devices do not implement class specifications, making it extremely difficult to write a single application code that can talk to all the devices.

3.4 Common-class API

Common-class API is a layer of routines that implements the common-class specification of the USB. This layer interacts with the host API layer function. It is difficult to say which routines belong to this layer. It is a deliberate design attempt to reuse routines to minimize the code size.

Routines inside the common-class layer take advantage of the fact that in USB all devices are enumerated with the same sequence of commands. When a USB device is plugged into the host hardware, it generates an interrupt, and the lower-level driver calls the common-class layer to start the device. Routines inside the common-class layer allocate memory, assign the USB address, and enumerate the device. After the device descriptors are identified, the common-class layer searches for applications that are registered for the class or device plug-in.

The following figure illustrates how device plug-in works.

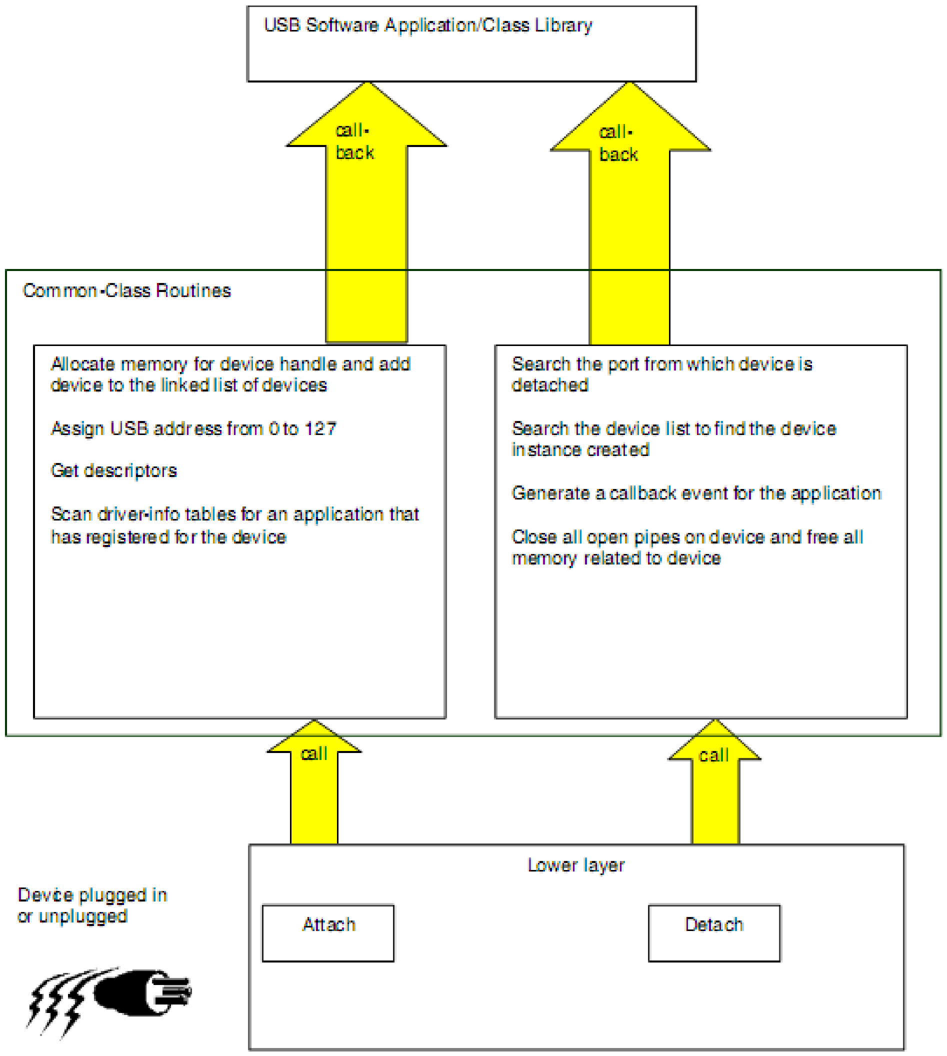

USB Software Application/Class Library
call-back
call-back
Common-Class Routines
Allocate memory for device handle and add device to the linked list of devices
Assign USB address from 0 to 127
Get descriptors
Scan driver-info tables for an application that has registered for the device
Search the port from which device is detached
Search the device list to find the device instance created
Generate a callback event for the application
Close all open pipes on device and free all memory related to device
call
call
Lower layer
Attach
Detach
Device plugged in or unplugged

Figure 3-2. How devices are attached and detached

3.5 USB Chapter 9 API

The USB specification document includes chapter 9, which is dedicated to a standard command protocol implemented by all USB devices. Every USB device is required to respond to a certain set of requests from the host. This is a low-level API that implements all USB chapter 9 commands. All customer applications can be written to use only this API without the common-class API or class libraries.

The USB chapter 9 commands are outside the scope of this document and require a good familiarity with USB protocol and higher level abstraction of USB devices. Here are some of the example routines that are implemented by this API. See the source code for implementation details.

- **usb_host_ch9_dev_req()** — For sending control pipe setup packets.
- **_usb_ch9_clear_fearture()** — For a clear feature USB command.
- **_usb_host_ch9_get_descriptor()** — For receiving descriptors from device.

3.6 Host API

The Host API is the hardware abstraction layer of the Freescale USB host. This layer implements routines independent of underlying USB controllers. For example, **usb_host_init()** initializes the host controller by calling the proper hardware-dependent routine. Similarly **usb_host_shutdown()** shuts down the proper hardware host controller. Here are the architecture functions implemented by this layer.

- Allocating pipes from a pool of pre-allocated pipe memory structures when **usb_host_open_pipe()** is called.
- Maintaining a list of all transfers pending per pipe. This is used in freeing memory when the pipe is closed.
- Maintaining a list of all services (callbacks) registered with the stack. When a specific hardware event such as attach and detach occurs, this layer generates a callback and informs the upper layers.
- Providing routines to cancel USB transfers by calling hardware-dependent functions.
- Providing other hardware-control routines such as the ability to shut down the controller, suspend the bus, and so on.

A good understanding of the source inside the API layer can be developed by reading the API routine and tracing it to the hardware drivers.

3.7 KHCI (Host Controller Interface)

KHCI is a completely hardware-dependent set of routines responsible for queuing and processing of USB transfers and searching for hardware events. Source understanding of this layer requires understanding of hardware.

Chapter 4 Developing Applications

4.1 Background

In the USB system, the host software controls the bus and talks to the target devices under the rules defined by the specification. A device is represented by a configuration that is a collection of one or more interfaces. Each interface comprises one or more endpoints. Each endpoint is represented as a logical pipe from the application software perspective.

The host application software registers for services with the USB host stack and describes the callback routines inside the driver info table. When a USB device is connected, the USB host stack driver enumerates the device automatically and generates interrupts for the application software. One interrupt per interface is generated on the connected device. If the application must talk to an interface, it can select that interface and receive the pipe handles for all the endpoints. Refer to *Freescale USB S tack with PHDC Host API Reference Manual* (Freescale document USBHOSTAPIRM) with the source code example to see what routines are called to find pipe handles. After the application software receives the pipe handles, it can start communication with the pipes. If the software must interact with another interface or configuration, it can call the appropriate routines again to select another interface.

The USB host stack is a few lines of code executed before starting communication with the USB device. Examples on the USB stack can be written with only a host API. However, most examples supplied with the stack are written using class drivers. Class drivers work with the host API as a supplement to the functionality. They make it easy to achieve the target functionality (see example sources for detail) without dealing with the implementation of standard routines. The following code steps are taken inside a host application driver for any specific device.

4.2 Developing an application

4.2.1 Create a project

Perform these steps to develop a new application:

1. To develop a new application, create an application sub-directory under /host/app directory.

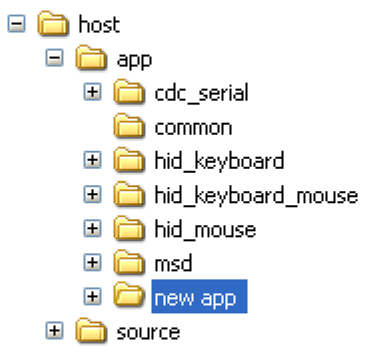


Figure 4-1. Create directory for application

2. Copy the usb_classes.h file from similar pre-existing applications. usb_classes.h is used to define classes that are used in an application.

   Example: If you want to create an application that uses HID and HUB, you can define them as:

```
#define USBCLASS_INC_HID
#define USBCLASS_INC_HUB
```

3. Create the CodeWarrior directory where the project files for a new application can be created.

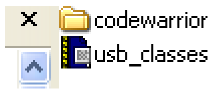


Figure 4-2. Create CodeWarrior project and usb_classes.h file

4. Create new files for creating the main application function and the callback.

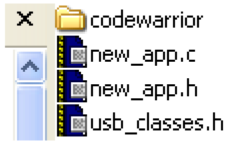


Figure 4-3. Create header and code file

— The new_app.h file contains application types and definitions:

Examples:

Define states of device:

```
/* Used to initialize USB controller */
#define MAX_FRAME_SIZE            1024
#define HOST_CONTROLLER_NUMBER       0

#define HID_BUFFER_SIZE                   4

#define  USB_DEVICE_IDLE                  (0)
#define  USB_DEVICE_ATTACHED              (1)
#define  USB_DEVICE_CONFIGURED            (2)
#define  USB_DEVICE_SET_INTERFACE_STARTED (3)
```

```
#define  USB_DEVICE_INTERFACED                  (4)
#define  USB_DEVICE_SETTING_PROTOCOL            (5)
#define  USB_DEVICE_INUSE                       (6)
#define  USB_DEVICE_DETACHED                    (7)
#define  USB_DEVICE_OTHER                       (8)

/*
** Following structs contain all states and pointers
** used by the application to control/operate devices.
*/

typedef struct device_struct {
   uint_32                          DEV_STATE;  /* Attach/detach state */
   _usb_device_instance_handle      DEV_HANDLE;
   _usb_interface_descriptor_handle INTF_HANDLE;
   CLASS_CALL_STRUCT                CLASS_INTF; /* Class-specific info */
} DEVICE_STRUCT,  _PTR_ DEVICE_STRUCT_PTR;

/* Alphabetical list of Function Prototypes */

#ifdef __cplusplus
extern "C" {
#endif

void usb_host_hid_recv_callback(_usb_pipe_handle, pointer, uchar_ptr,
uint_32,
   uint_32);
void usb_host_hid_ctrl_callback(_usb_pipe_handle, pointer, uchar_ptr,
uint_32,
   uint_32);
void usb_host_hid_mouse_event(_usb_device_instance_handle,
   _usb_interface_descriptor_handle, uint_32);

#ifdef __cplusplus
}
#endif
```

— The new_app.c file contains driver informations, callback functions, event functions, and main function.

4.2.2 Define a driver info table

A driver information table defines devices that are supported and handled by this target application. This table defines the PID, VID, class, and subclass of the USB device. The host/device stack generates an attached callback when a device matches this table entry. The application now can communicate with the device. The following structure defines one member of the table. If the Vendor-Product pair does not match for a device, Class, Subclass, and Protocol are checked to match. Use 0xFF in Subclass and Protocol struct member to match any Subclass/Protocol.

```
/* Information for one class or device driver */
typedef struct driver_info
{
   uint_8          idVendor[2];      /* Vendor ID per USB-IF */
   uint_8          idProduct[2];     /* Product ID per manufacturer */
   uint_8          bDeviceClass;     /* Class code, 0xFF if any */
   uint_8          bDeviceSubClass;  /* Sub-Class code, 0xFF if any */
   uint_8          bDeviceProtocol;  /* Protocol, 0xFF if any */
   uint_8          reserved;         /* Alignment padding */
event_callback attach_call;          /* event callback function*/
} USB_HOST_DRIVER_INFO, _PTR_ USB_HOST_DRIVER_INFO_PTR;
```

The following is a sample driver info table. See the example source code for samples. Note the following table defines all HID MOUSE devices that are boot subclasses. A terminating NULL entry in the table is always created for search end.

Because two classes (HID and HUB) are used in the HID MOUSE application, the DriverInfoTable variable has three elements. There are two event callback functions for two classes: **usb_host_hid_keyboard_event** for HID class and **usb_host_hub_device_event** for HUB class.

```
   /* Table of driver capabilities this application wants to use */
   static USB_HOST_DRIVER_INFO DriverInfoTable[] = {
       {
               {0x00, 0x00},        /* Vendor ID per USB-IF */
               {0x00, 0x00},        /* Product ID per manufacturer */
               USB_CLASS_HID,       /* Class code */
               USB_SUBCLASS_HID_BOOT, /* Sub-Class code */
               USB_PROTOCOL_HID_KEYBOARD, /* Protocol */
               0,                   /* Reserved */
   <add the name of your event callback function here>
               usb_host_hid_keyboard_event /* Application call back function */
           },
       /* USB 1.1 hub */
       {
               {0x00, 0x00},        /* Vendor ID per USB-IF */
               {0x00, 0x00},        /* Product ID per manufacturer */
               USB_CLASS_HUB,       /* Class code */
               USB_SUBCLASS_HUB_NONE, /* Sub-Class code */
               USB_PROTOCOL_HUB_LS, /* Protocol */
               0,                   /* Reserved */
   <add the name of your event callback function here>
               usb_host_hub_device_event /* Application call back function */
           },
       {
               {0x00, 0x00},        /* All-zero entry terminates */
               {0x00, 0x00},        /* driver info list. */
               0,
               0,
               0,
               0,
           NULL},
   }
```

4.2.3 Main application function flow

In the main application function, it is necessary to follow these steps:

1. Initializing hardware
2. Initializing the host controller
3. Registering service
4. Calling tasks in a forever loop

4.2.3.1 Initializing hardware

The first step to run an application is hardware initialization. It is necessary to initialize the processor (mode and clock), SCI (to use UART), and real-time clock (to use some delay functions).

4.2.3.2 Initializing the host controller

The second step required to act as a host is to initialize the stack in a host mode. This allows the stack to install a host interrupt handler and initialize the necessary memory required to run the stack. The following example illustrates this:

```
status = _usb_host_init(HOST_CONTROLLER_NUMBER, /* Use value in header file */
        MAX_FRAME_SIZE,         /* Frame size per USB spec  */
       &host_handle);          /* Returned pointer */
```

The second argument (MAX_FRAME_SIZE) in the above code is the size of the periodic frame list. Full speed customers can ignore this argument.

4.2.3.3 Register services

Once the host is initialized, the USB host stack is ready to provide services. An application can register for services as documented in *Freescale USB Stack Host API Reference Manual* (document USBHOSTAPIRM). The host API document describes how the application is registered for this device because the driver info table already registers a callback routine. The following example shows how to register for a service on the host stack:

```
/*
     ** since we are going to act as the host driver, register the driver
     ** information for wanted class/subclass/protocols
     */
    status = _usb_host_driver_info_register(host_handle, DriverInfoTable);
    if(status != USB_OK) {
        printf("\nDriver Registration failed. STATUS: %x", status);
        fflush(stdout);
        exit(1);
    }
```

4.2.3.4 Run the process task

The last step in the main application function is to call **_usb_khci_task()** and the application task. These tasks are called in the forever loop.

```
    for(;;) {
        <User call the application task here>
        _usb_khci_task();
        __RESET_WATCHDOG(); /* feeds the dog */
      } /* loop forever */
```

4.2.4 Event callback function

After the software has registered the driver info table and register for other services, it is ready to handle devices. In the USB Host stack, customers do not have to write any enumeration code. As soon as the device is connected to the host controller, the USB Host stack enumerates the device and finds how many interfaces are supported. Also, for each interface it scans the registered driver info tables and finds which application has registered for the device. It provides a callback if the device criteria matched the table. The application software has to choose the interface. You can implement the event callback function as follows:

```
void usb_host_hid_keyboard_event(
    /* [IN] pointer to device instance */
    _usb_device_instance_handle dev_handle,
    /* [IN] pointer to interface descriptor */
    _usb_interface_descriptor_handle intf_handle,
    /* [IN] code number for event causing callback */
    uint_32 event_code)
{
    INTERFACE_DESCRIPTOR_PTR intf_ptr = (INTERFACE_DESCRIPTOR_PTR) intf_handle;
    switch (event_code) {
        case USB_ATTACH_EVENT:
        case USB_CONFIG_EVENT:
        <Add your code here>
        break;

        case USB_INTF_EVENT:
        <Add your code here>
        break;

        case USB_DETACH_EVENT:
        <Add your code here>
        break;
        }
}
```

Here is sample code for the HID MOUSE application. In this code, the hid_device variable contains all states and pointers used by the application to control/operate the device:

```
void usb_host_hid_keyboard_event(
    /* [IN] pointer to device instance */
    _usb_device_instance_handle dev_handle,
    /* [IN] pointer to interface descriptor */
    _usb_interface_descriptor_handle intf_handle,
    /* [IN] code number for event causing callback */
    uint_32 event_code)
{
    INTERFACE_DESCRIPTOR_PTR intf_ptr = (INTERFACE_DESCRIPTOR_PTR) intf_handle;
    fflush(stdout);
    switch (event_code) {

    case USB_ATTACH_EVENT:
        printf("----- Attach Event -----\r\n");
        /* Drop through config event for the same processing */
    case USB_CONFIG_EVENT:
        printf("State = %d", hid_device.DEV_STATE);
        printf("  Class = %d", intf_ptr->bInterfaceClass);
        printf("  SubClass = %d", intf_ptr->bInterfaceSubClass);
        printf("  Protocol = %d\r\n", intf_ptr->bInterfaceProtocol);
        fflush(stdout);

        if(hid_device.DEV_STATE == USB_DEVICE_IDLE) {
            hid_device.DEV_HANDLE = dev_handle;
            hid_device.INTF_HANDLE = intf_handle;
            hid_device.DEV_STATE = USB_DEVICE_ATTACHED;
        }
        else {
            printf("HID device already attached\r\n");
            fflush(stdout);
        }
        break;

    case USB_INTF_EVENT:
        printf("----- Interfaced Event -----\r\n");
        hid_device.DEV_STATE = USB_DEVICE_INTERFACED;
        break;

    case USB_DETACH_EVENT:
        /* Use only the interface with desired protocol */
        printf("\r\n----- Detach Event -----\r\n");
        printf("State = %d", hid_device.DEV_STATE);
        printf("  Class = %d", intf_ptr->bInterfaceClass);
        printf("  SubClass = %d", intf_ptr->bInterfaceSubClass);
        printf("  Protocol = %d\r\n", intf_ptr->bInterfaceProtocol);
        fflush(stdout);

        hid_device.DEV_HANDLE = NULL;
        hid_device.INTF_HANDLE = NULL;
        hid_device.DEV_STATE = USB_DEVICE_DETACHED;
        break;
    }

    /* notify application that status has changed */
    _usb_event_set(&USB_Event, USB_EVENT_CTRL);
}
```

NOTE

The example shows how a mouse can be connected to the host directly or using a hub. However, the hid_device global structure limits this usage to one single device. It will not work with multiple devices connected through the hub.

4.2.5 Selecting an interface on the device

If the interface handle has been obtained, application software can select the interface that a retrieve pipe handles. The following code demonstrates this procedure:

```
case USB_DEVICE_ATTACHED:
    printf("Keyboard device attached\n");
    hid_device.DEV_STATE = USB_DEVICE_SET_INTERFACE_STARTED;
    status = _usb_hostdev_select_interface(hid_device.DEV_HANDLE,
    hid_device.INTF_HANDLE, (pointer) & hid_device.CLASS_INTF);
    if(status != USB_OK) {
        printf("\nError in _usb_hostdev_select_interface: %x", status);
        fflush(stdout);
        exit(1);
    }
    break;
```

As internal information, **usb_hostdev_select_interface** caused the stack to allocate memory and to do the necessary preparation to start communicating with this device. This routine opens logical pipes and allocates bandwidths on periodic pipes. This allocation of bandwidths can be time-consuming under complex algorithms.

4.2.6 Retrieving and storing pipe handles

If the interface has been selected, pipe handles can be retrieved by calling as shown in this example:

```
pipe = _usb_hostdev_find_pipe_handle(hid_device.DEV_HANDLE,
hid_device.INTF_HANDLE, USB_INTERRUPT_PIPE, USB_RECV);
```

In this code, pipe is a memory pointer that stores the handle (see code example for details). Note that this routine specified the type of pipe retrieved. The code shows how to communicate with a mouse that has an interrupt to obtain the pipe handle for the interrupt pipe.

4.2.7 Sending/receiving data to/from device

The USB packet transfers on the USB software function in terms of transfer requests (TR). A similar term in Windows and Linux is URB. In Windows, drivers keep sending URBs down the stack and wait for events or callbacks for USB completion. There is one callback or event per URB completion. The USB stack concept is the same except that the fields inside a TR can be different. A TR is a memory structure that describes a transfer in its entirety. The USB stack provides a helper routine called **usb_hostdev_tr_init()** that can be used to initialize a TR. Every TR down the stack has a unique number assigned by the **tr_init()** routine. The following code example shows how this routine is called:

```
usb_hostdev_tr_init(&tr, usb_host_hid_recv_callback, NULL);
```

The routine takes the `tr` pointer to the structure that needs to be initialized and the name of the callback routine that is called when this TR completes. An additional parameter can be supplied that is called back when TR completes. The user can throw away TR immediately after it was used in _usb_host_recv_data. The reason is that TR is copied to the pipe handle. The user does not need to save all the information because a copy is already used for the USB stack, so perhaps the best method is to allocate TRs on the stack.

After TR is initialized and the pipe handle is available, it is easy to send and receive data to the device. USB devices that use periodic data need a periodic call to send or receive data. It is recommended to use timers to ensure that a receive or send data call is done in a timely manner, so the packets to and from the device are not lost. These USB driver design details are outside the scope of this document. The following code provides an example of how the data is received.

```
      status = _usb_host_recv_data(host_handle, pipe, &tr);
```

Chapter 5 FAT File System

5.1 Introduction

The FATFS module is developed based on MSD host class of Freescale USB Stack Software Suite. Its architecture contains USB driver code, disk I/O interface functions, FAT APIs, and some applications. This document intends to help you gain an insight into the File Allocation Table and capabilities to develop your own applications. The document is targeted for firmware application developers who would like to develop the applications using FATFS file system module.

5.2 File Allocation Table Overview

The mass storage media is organized logically as a Master Boot Record and several partitions. Figure 5-1 describes the logical structures of a mass storage medium.

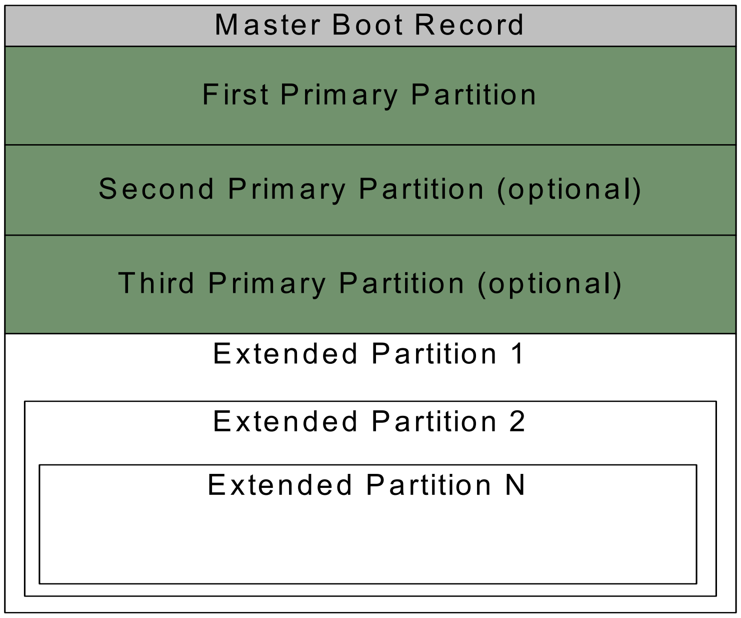


Figure 5-1. Logical structure of mass storage media

The Master Boot Record is located at sector zero. It contains three items: an area for executable code, a partition table, and a boot signature. The partition table enables defining one or more partitions, or logical volumes, in the storage media. Many devices have just one volume. The partition table in the MBR sector has room for four 16-byte entries that specify the sectors that belong to a partition.

A FAT partition composed of four different sections as shown in the following figure.

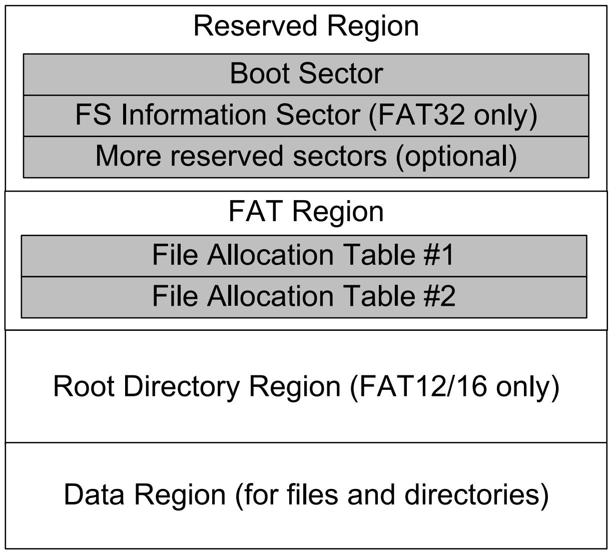

Reserved Region
Boot Sector
FS Information Sector (FAT32 only)
More reserved sectors (optional)
FAT Region
File Allocation Table #1
File Allocation Table #2
Root Directory Region (FAT12/16 only)
Data Region (for files and directories)

Figure 5-2. FAT partition structure

- **Reserved Region** These sectors are located at the very beginning. The first reserved sector (sector 0) is the Boot Sector (*Partition Boot Record*). It includes an area called the BIOS Parameter Block (with some basic file system information, in particular its type, and pointers to the location of the other sections) and usually contains the operating system's boot loader code. The total count of reserved sectors is indicated by a field inside the Boot Sector. For FAT32 file systems, the reserved sectors include a *File System Information Sector* at Sector 1 and a *Backup Boot Sector* at Sector 6.
- **FAT Region**. This typically contains two copies of the *File Allocation Table* for the sake of redundancy checking, although the extra copy is rarely used, even by disk repair utilities. These are maps of the Data Region, indicating which clusters are used by files and directories. In FAT16 and FAT12, they immediately follow the reserved sectors.
- **Root Directory Region**. This is a *Directory Table* that stores information about the files and directories located in the root directory. It is only used with FAT12 and FAT16, and imposes on the root directory a fixed maximum size which is pre-allocated at creation of this volume. FAT32 stores the root directory in the Data Region, along with files and other directories, allowing it to grow without such a constraint. Therefore, for FAT32, the Data Region starts here.
- **Data Region**. This is where the actual file and directory data is stored and takes up most of the partition. The size of files and subdirectories can be increased arbitrarily (as long as there are free clusters) by simply adding more links to the file's chain in the FAT. Note that the files are allocated in units of clusters, so if a 1 KB file resides in a 32 KB cluster, 31 KB are wasted. FAT32 typically commences the Root Directory Table in cluster number 2, the first cluster of the Data Region.

FAT uses little endian format for entries in the header and the FAT(s).

5.3 Software Module

5.3.1 USB FATFS Feature

The USB FATFS software module uses class MSD's APIs of Freescale USB Stack Host to access mass storage device. The module supports:

- FAT sub-types: FAT12, FAT16, and FAT32
- Number of open files: Unlimited, depends on available memory
- Multi-partition: Number of volumes (up to 10)
- File size: Depends on FAT specs (up to 4 GB)
- Volume size: Depends on FAT specs (up to 2 TB on 512 bytes/sector)
- Cluster size: Depends on FAT specs (up to 64 KB on 512 bytes/sector)
- Sector size: Depends on FAT specs (up to 4 KB)
- Long file name support in ANSI/OEM or Unicode
- Multiple ANSI/OEM code pages including DBCS
- Code size reduction depending on user configuration

The class drivers are programmed with generic code, so they can be used with other processors if standard SCSI commands are provided like MSD class of the Freescale USB Host Stack.

5.3.2 Module license

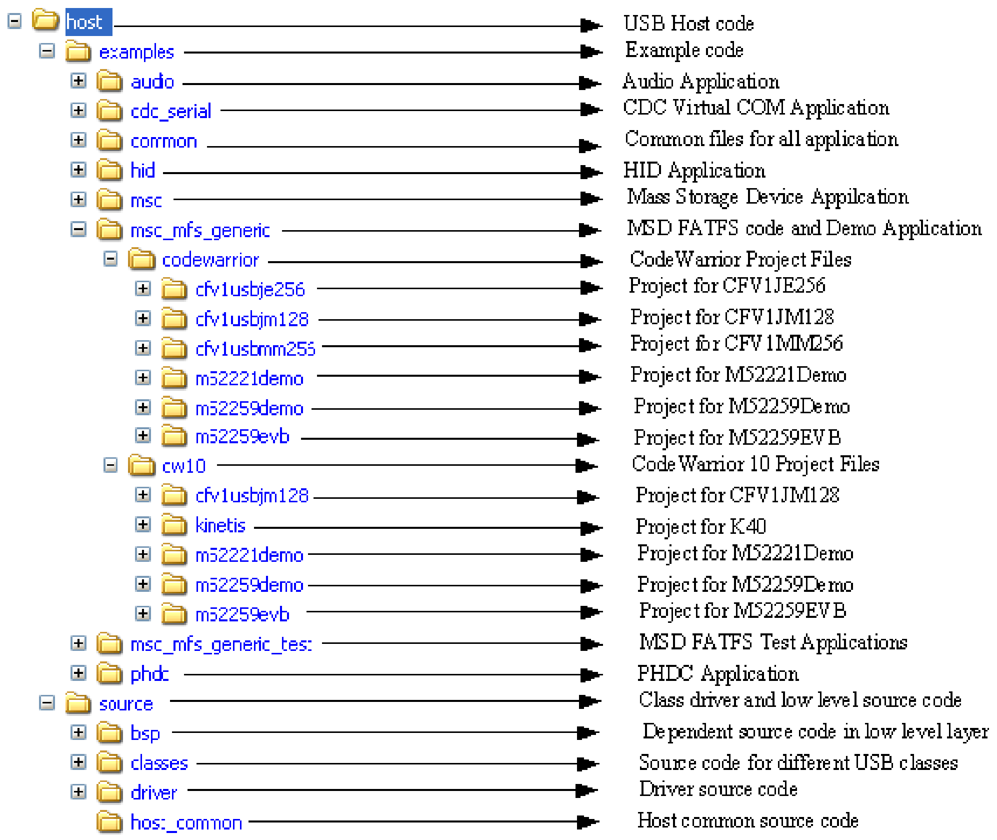
FATFS is an open source module. It follows the BSD-style license. Redistributions of source code must retain the following copyright notice.

Because, FATFS is for embedded projects, the conditions for redistributions in binary form, such as embedded code, hex file, and binary library are not specified to increase its usability. The documentation of the distributions need not include FATFS and its license notice.

5.4 Directory structure

The software module has a standard directory structure. You can extend it easily to accommodate more applications for different processor families.

Figure 5-3 shows the directory structure:



```
host                        USB Host code
  examples                  Example code
    audio                   Audio Application
    cdc_serial              CDC Virtual COM Application
    common                  Common files for all application
    hid                     HID Application
    msc                     Mass Storage Device Appilcation
    msc_mfs_generic         MSD FATFS code and Demo Application
      codewarrior           CodeWarrior Project Files
        cfv1usbje256        Project for CFV1JE256
        cfv1usbjm128        Project for CFV1JM128
        cfv1usbmm256        Project for CFV1MM256
        m52221demo          Project for M52221Demo
        m52259demo          Project for M52259Demo
        m52259evb           Project for M52259EVB
      cw10                  CodeWarrior 10 Project Files
        cfv1usbjm128        Project for CFV1JM128
        kinetis             Project for K40
        m52221demo          Project for M52221Demo
        m52259demo          Project for M52259Demo
        m52259evb           Project for M52259EVB
    msc_mfs_generic_test    MSD FATFS Test Applications
    phdc                    PHDC Application
  source                    Class driver and low level source code
    bsp                     Dependent source code in low level layer
    classes                 Source code for different USB classes
    driver                  Driver source code
    host_common             Host common source code
```

Figure 5-3. MSD FATFS with Freescale USB Host Stack directory structure

5.5 USB FATFS Architecture

This chapter provides an overview of USB FATFS architecture and its software flow.

5.6 Architecture overview

The architecture of USB FATFS is shown in the following figure.

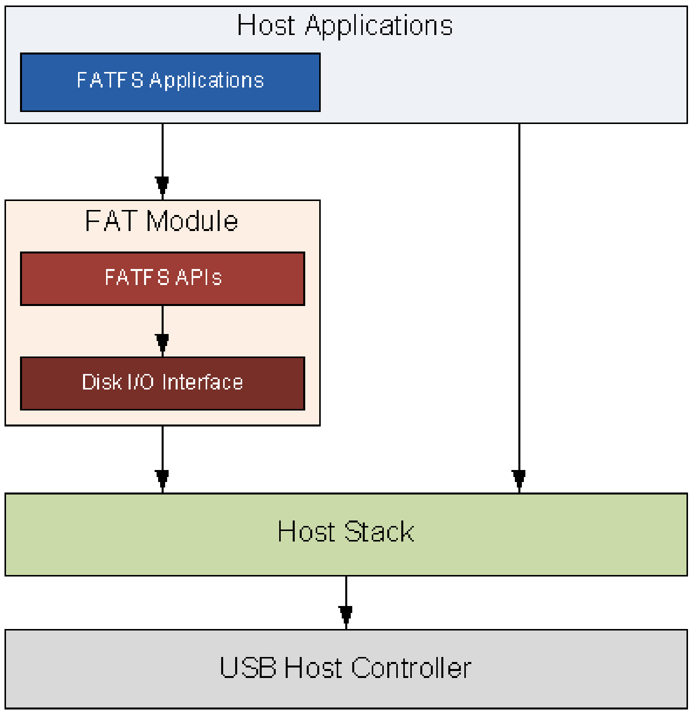


Figure 5-4. FATFS Architecture

The remainder of the document describes only the FATFS module. For more information about the host stack structure and functionality or about the demo application for the different USB classes, refer to the *Freescale USB Stack Host Users Guide* (document USBHOSTUG).

5.7 FATFS Module overview

5.7.1 FATFS APIs

The FATFS APIs layer implements file system APIs such as f_open, f_read, f_writes, and so on. This layer is independent with USB Host Stack. It uses Disk I/O interface to communicate with mass storage device. The set of APIs is divided into four groups:

1. Group of APIs that operates with logical volume or partition.
2. Group of APIs that operates with directory.
3. Group of APIs that operates with file.
4. Group of APIs that operates with both file and directory.

APIs of FATFS are listed in Section 5.8.2, "Configuration options."

5.7.2 Disk I/O interface

The Disk I/O Interface consists of six APIs that are used by FATFS API to access and manage data in mass storage device. To confirm with FATFS APIs, the functions must follow the prototype described in section **Disk I/O Interface** of ***FatFs Generic File System Module*** document. The layer operates with USB Host

Stack via three SCSI commands: READ10, WRITE10, and READ CAPACITY that are implemented on Host stack.

The following table lists the APIs of Disk I/O Interface layer.

Table 5-1. Disk I/O interface APIs

APIs	Descriptions
disk_initialize	Initialize disk drive
disk_status	Get disk status
disk_read	Read data sector(s) from mass storage device
disk_write	Write data sector(s) to mass storage device
disk_ioctl	Get information about sector size, sector count and physical volume size.
get_fattime	Get current time of system. At this time, system time utility has not been implemented, so that the function always returns to a fixed date.

5.8 Developing Applications for FATFS

5.8.1 Background

FATFS module contains various configuration options. Therefore, this chapter provides information to help user select proper options depending on his requirement to reach the highest performance. Moreover, how to create a new FATFS project is also mentioned here.

5.8.2 Configuration options

The following table shows the options for module size reduction.

Table 5-2. Module size reduction options

API	_FS_MINIMIZE				_FS_READ ONLY		_USE_STR FUNC		_FS_RPATH			_USE_MKF S		_USE_FOR WARD	
	0	1	2	3	0	1	0	1	0	1	2	0	1	0	1
f_mount															
f_open															
f_close															
f_read															
f_write						x									
f_sync						x									
f_lseek				x											
f_opendir			x	x											
f_readdir			x	x											

Table 5-2. Module size reduction options

API	_FS_MINIMIZE				_FS_READ ONLY		_USE_STR FUNC		_FS_RPATH			_USE_MKF S		_USE_FOR WARD	
	0	1	2	3	0	1	0	1	0	1	2	0	1	0	1
f_stat		x	x	x											
f_getfree		x	x	x		x									
f_truncate		x	x	x		x									
f_unlink		x	x	x		x									
f_mkdir		x	x	x		x									
f_chmod		x	x	x		x									
f_utime		x	x	x		x									
f_remane		x	x	x		x									
f_chdir									x						
f_chdrive									x						
f_getcwd									x	x		x			
f_mkfs						x									
f_forward														x	
f_putc						x	x								
f_puts						x	x								
f_printf						x	x								
f_gets							x								
f_eof															
f_error															
f_tell															
f_size															

x- API is removed

Other configuration options for FATFS module are described in the following table.

Table 5-3. General FATFS configuration options

Feature	Option	Value	Description
Multi-partitions	_VOLUMES	1 to 4	Number of volumes to be used
	_MULTI_PARTITION	0	Disable multi-partitions feature
		1	Enable multi-partitions feature
Memory access	_WORD_ACCESS	0	Retrieve data from FAT volume byte by byte
		1	Retrieve data from FAT volume word by word
Open multi-files	_FS_SHARE	integer	Number of files can be opened simultaneously for write
Memory size	_FS_TINY	0	FATFS uses the sector buffer in the system for file data transfer. This reduces memory consumption 512 bytes each file object
		1	FATFS uses a sector buffer for the individual file object for file data transfer
Sector size	_MAX_SS	512, 1024, 2048, 4096	Maximum sector size to be handled

Table 5-3. General FATFS configuration options (continued)

Feature	Option	Value	Description
Long File Name	_CODE_PAGE	437	Used U.S. (OEM)
		720	Used Arabic (OEM)
		737	Used Greek (OEM)
		775	Used Baltic (OEM)
		850	Used Multilingual Latin 1 (OEM)
		858	Used Multilingual Latin 1 + Euro (OEM)
		852	Used Latin 1 (OEM)
		855	Used Cyrillic (OEM)
		866	Used Russian (OEM)
		857	Used Turkish (OEM)
		862	Used Hebrew (OEM)
		874	Used Thai (OEM, Windows)
		1	ASCII only (valid for non - LFN configuration)
		1250	Used Central Europe (Windows)
		1251	Used Cyrillic (Windows)
		1252	Used Latin 1 (Windows)
		1253	Used Greek (Windows)
		1254	Used Turkish (Windows)
		1255	Used Hebrew (Windows)
		1256	Used Arabic (Windows)
		1257	Used Baltic (Windows)
		1278	Used Vietnam (OEM, Windows)
	_USE_LFN	0	Disable LFN feature. _MAX_LFN and _LFN_UNICODE have no effect
		1	Enable LFN with static working buffer on the BSS
		2	Enable LFN with dynamic working buffer on the STACK
		3	Enable LFN with dynamic working buffer on the HEAP
	-MAX_LFN	12 to 255	Maximum LFN length to handle
	_LFN_UNICODE	0	The character code set on FATFS APIs is ANSI/OEM
		1	The character code set on FATFS APIs is Unicode
	_FS_RPATH	0	Disable relative path
		1	Enable relative path

Table 5-3. General FATFS configuration options (continued)

Feature	Option	Value	Description
Multi-partitions	**_VOLUMES**	1 to 4	Number of volumes to be used
	_MULTI_PARTITION	0	Disable multi-partitions feature
		1	Enable multi-partitions feature
Memory access	**_WORD_ACCESS**	0	Retrieve data from FAT volume byte by byte
		1	Retrieve data from FAT volume word by word
Open multi-files	**_FS_SHARE**	integer	Number of files can be opened simultaneously for write
Memory size	**_FS_TINY**	0	FATFS uses the sector buffer in the system for file data transfer. This reduces memory consumption 512 bytes each file object
		1	FATFS uses a sector buffer for the individual file object for file data transfer
Sector size	**_MAX_SS**	512, 1024, 2048, 4096	Maximum sector size to be handled

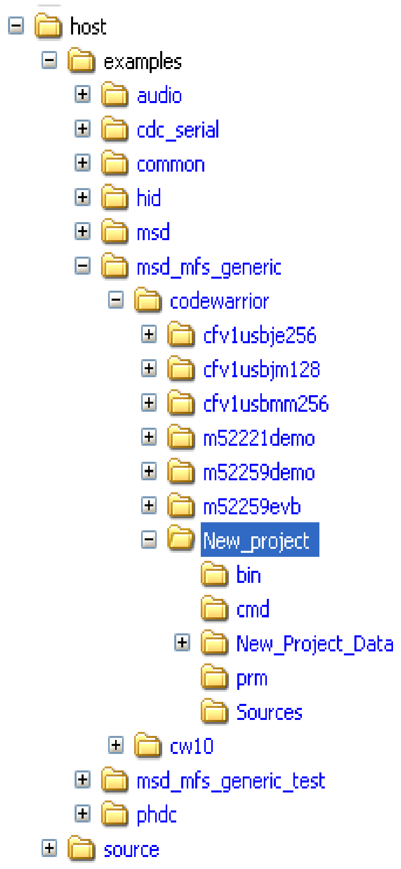
NOTE

Some USB sticks may pass check_fs() function because the boot signature in the BIOS parameter block may differ from 0xAA55 (offset 0x1FE) and also because "FAT" string differs for FAT12/16 (offset 0x36) or for FAT32 (0x52). This is an indication that the file system was not properly formatted to FAT.

5.8.3 Create a project

Perform these steps to develop a new application:

1. Create a new project under **/host/examples/msd_mfs_generic/codewarrior** or **/host/examples/msd_mfs_generic/cw10** directory.



```
⊟ host
   ⊟ examples
      ⊞ audio
      ⊞ cdc_serial
      ⊞ common
      ⊞ hid
      ⊞ msd
      ⊟ msd_mfs_generic
         ⊟ codewarrior
            ⊞ cfv1usbje256
            ⊞ cfv1usbjm128
            ⊞ cfv1usbmm256
            ⊞ m52221demo
            ⊞ m52259demo
            ⊞ m52259evb
            ⊟ New_project
                  bin
                  cmd
               ⊞ New_Project_Data
                  prm
                  Sources
         ⊞ cw10
      ⊞ msd_mfs_generic_test
      ⊞ phdc
   ⊞ source
```

Figure 5-5. Create a new project

2. Add **ccsbcs.h, diskio.h, diskio.c, ff.h, ff.c, ffconf.h, main.c, usb_class.h, msd_fat_demo.c**, and other files to the created project similar to the pre-existing FATFS applications.
3. Modify FATFS module options in the file **ffconf.h**.
4. Modify FATFS application task in the file **msd_fat_demo.c** (**fat_demo** function) as you want.

Appendix A Working with the Software

A.1 Introduction

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

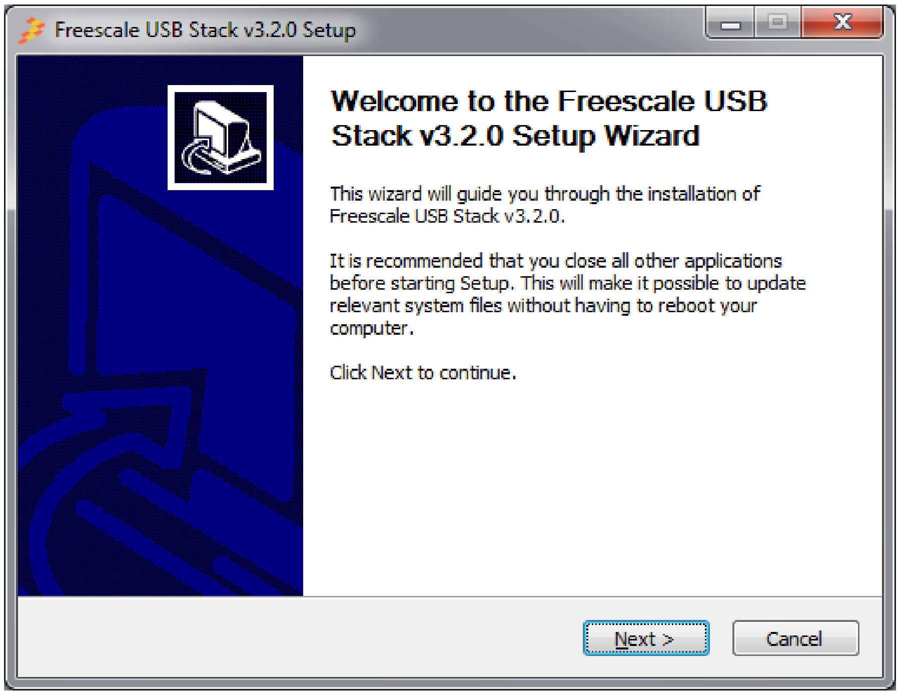
- Preparing the setup
- Building the application
- Running the application

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

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.
2. The Freescale USB Stack Setup window appears as shown in the following figure. Click on the **Next** button to continue.



Freescale USB Stack v3.2.0 Setup

Welcome to the Freescale USB Stack v3.2.0 Setup Wizard

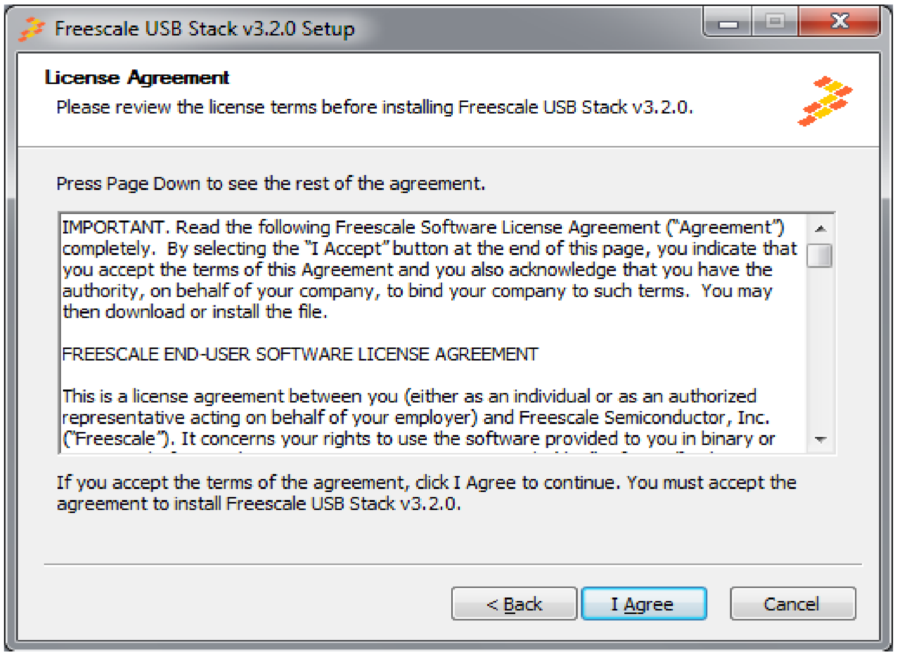
This wizard will guide you through the installation of Freescale USB Stack v3.2.0.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

Figure A-1. Freescale USB Stack Setup Wizard

3. In the following figure, click on the **I Agree** button to accept the license agreement.



Freescale USB Stack v3.2.0 Setup

License Agreement

Please review the license terms before installing Freescale USB Stack v3.2.0.

Press Page Down to see the rest of the agreement.

IMPORTANT. Read the following Freescale Software License Agreement ("Agreement") completely. By selecting the "I Accept" button at the end of this page, you indicate that you accept the terms of this Agreement and you also acknowledge that you have the authority, on behalf of your company, to bind your company to such terms. You may then download or install the file.

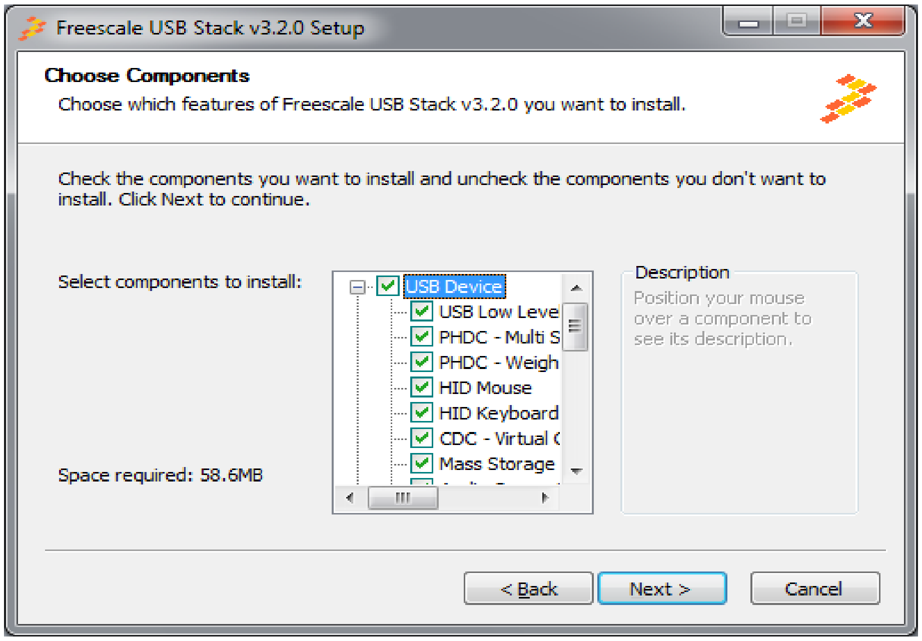
FREESCALE END-USER SOFTWARE LICENSE AGREEMENT

This is a license agreement between you (either as an individual or as an authorized representative acting on behalf of your employer) and Freescale Semiconductor, Inc. ("Freescale"). It concerns your rights to use the software provided to you in binary or

If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install Freescale USB Stack v3.2.0.

Figure A-2. Freescale USB Stack setup license agreement

4. In the following figure, select USB low level stack and other class components to install and click on the **Next** button.



Freescale USB Stack v3.2.0 Setup

Choose Components

Choose which features of Freescale USB Stack v3.2.0 you want to install.

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select components to install:

- ✓ USB Device
  - ✓ USB Low Leve
  - ✓ PHDC - Multi S
  - ✓ PHDC - Weigh
  - ✓ HID Mouse
  - ✓ HID Keyboard
  - ✓ CDC - Virtual C
  - ✓ Mass Storage

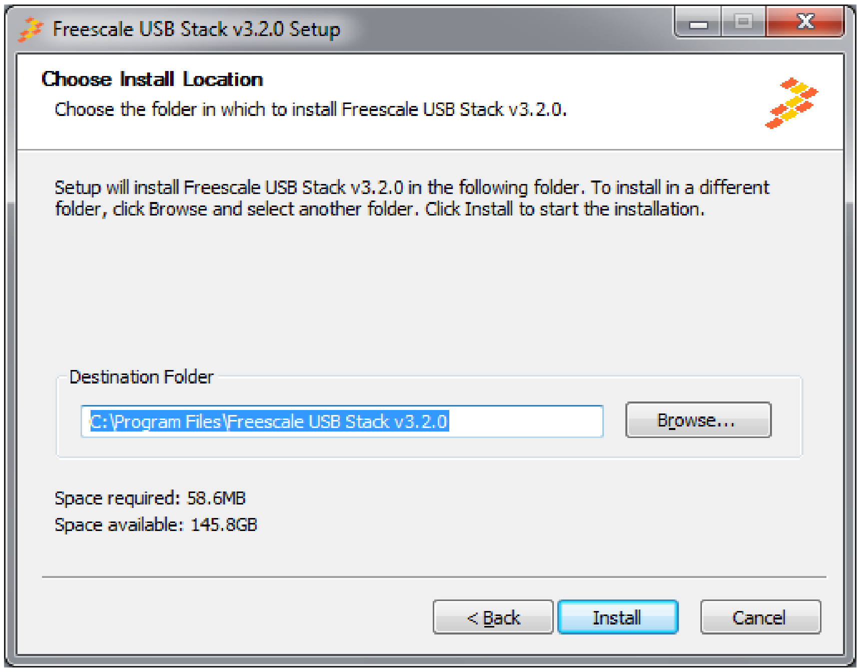
Space required: 58.6MB

Description

Position your mouse over a component to see its description.

Figure A-3. Freescale USB Stack components

5. In the following figure, select the location of the folder where you want to install the Freescale USB Stack software and click on the **Install** button.



Choose Install Location

Choose the folder in which to install Freescale USB Stack v3.2.0.

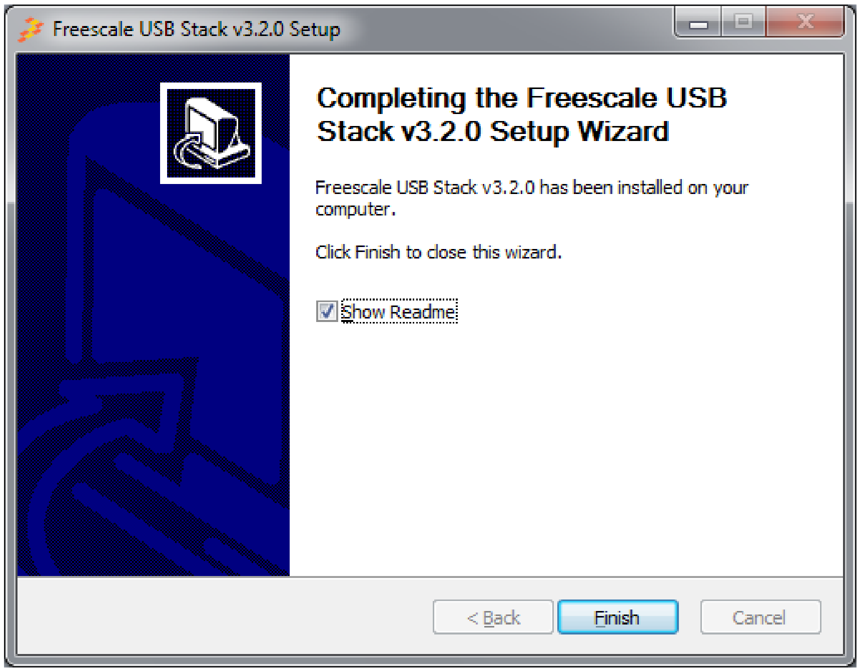
Setup will install Freescale USB Stack v3.2.0 in the following folder. To install in a different folder, click Browse and select another folder. Click Install to start the installation.

Destination Folder: C:\Program Files\Freescale USB Stack v3.2.0

Space required: 58.6MB
Space available: 145.8GB

Figure A-4. Freescale USB Stack installation folder location

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



Completing the Freescale USB Stack v3.2.0 Setup Wizard

Freescale USB Stack v3.2.0 has been installed on your computer.

Click Finish to close this wizard.

☑ Show Readme

Figure A-5. Freescale USB Stack installation finish

7. Click **Start** → **Programs** → **Freescale USB Stack** → **Source** → **USB Host** to launch the project.

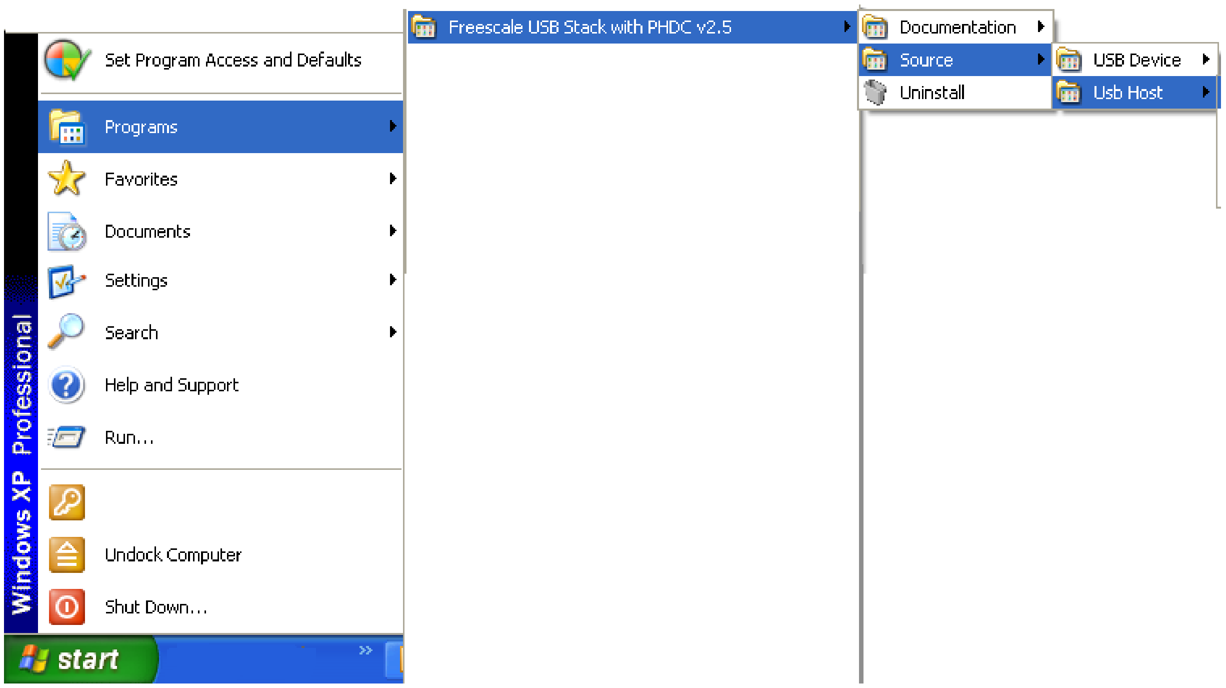


Figure A-6. Freescale USB Stack source program for launch

A.1.1.2 Hardware setup

Set up the connections as shown in the following figure.

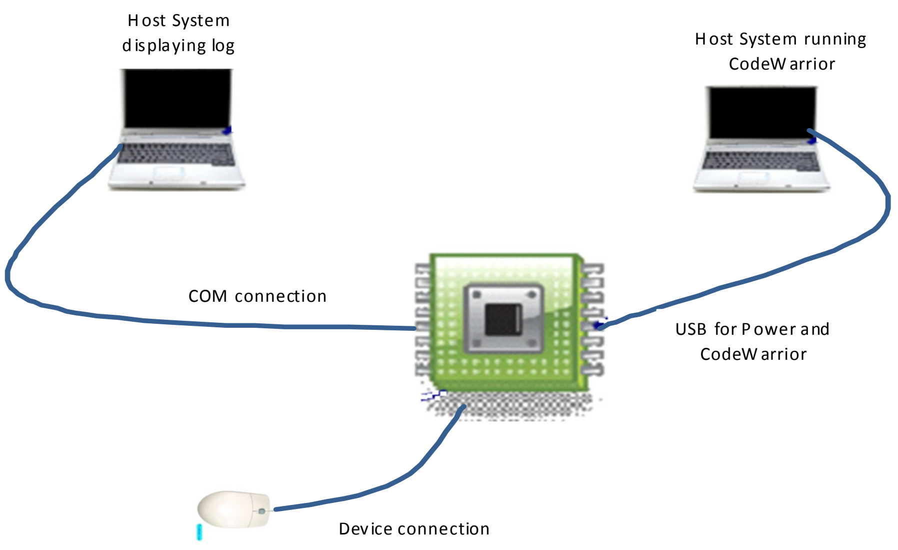

Host System displaying log
Host System running CodeWarrior
COM connection
USB for Power and CodeWarrior
Device connection

Figure A-7. Coldfire V1 and V2 USB setup

1. 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 for downloading the image to the flash.
2. Make the second connection between the DemoJM board and the personal computer, to display the log of DemoJM.
3. Make the third connection between the device and DemoJM.

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

The host software for CFV1 is built with CodeWarrior 6.3. In addition, the host 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 ColdFire V1 project:

1. Navigate to the project file and open cfv1jm128_xxx.mcp project file in CodeWarrior IDE.

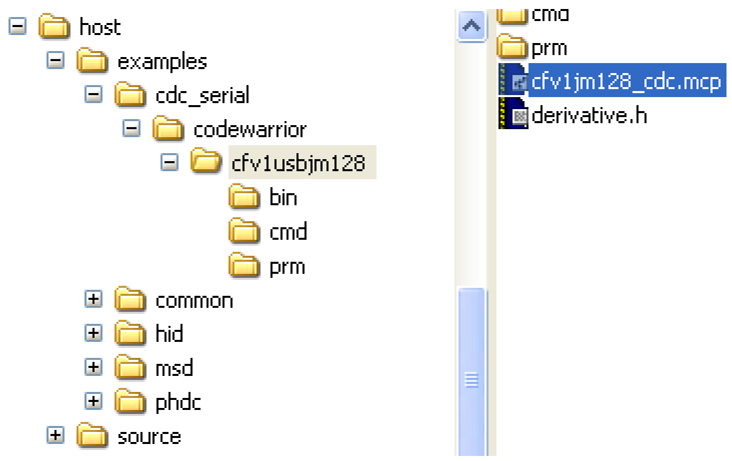
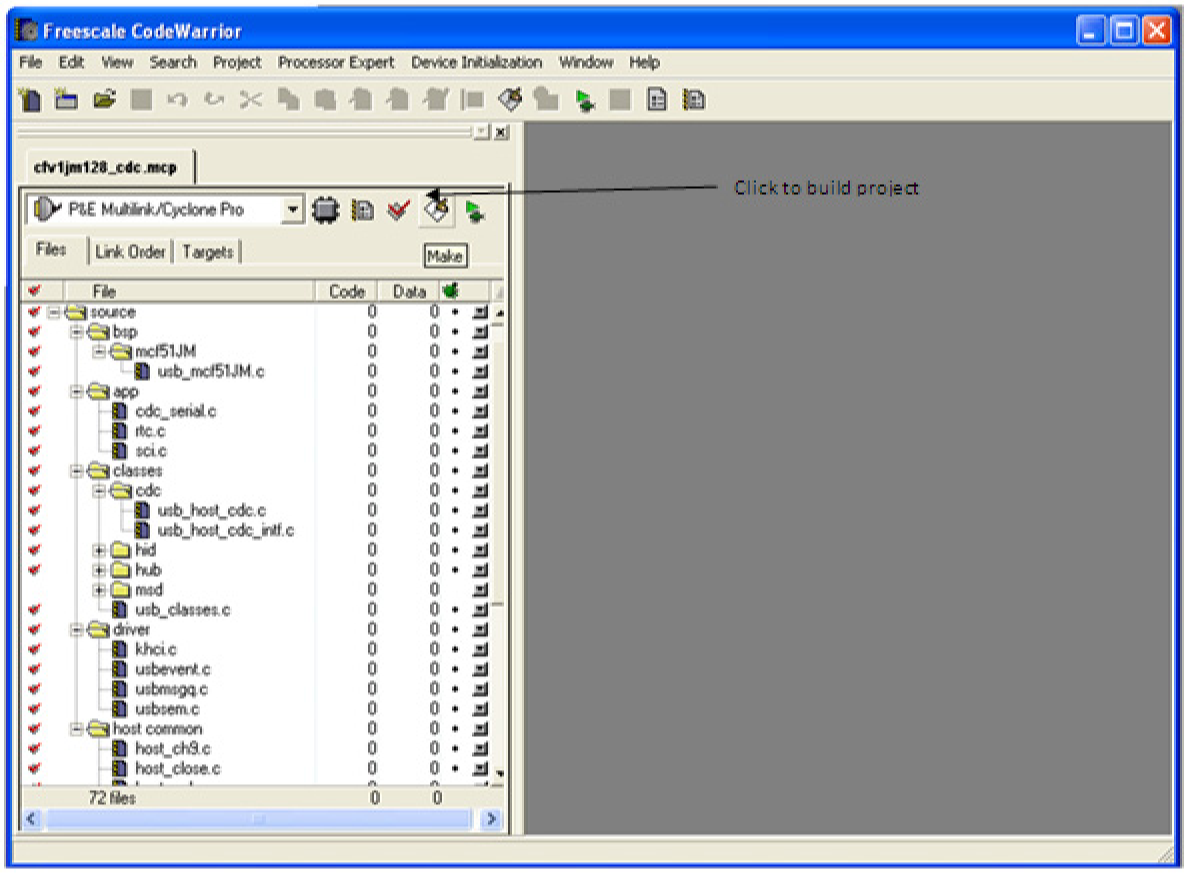


Figure A-8. Open cfv1jm128_xxx.mcp project file

2. After you have opened the project, the following window appears. To build the project, click the button as shown in the following figure.


Freescale CodeWarrior
File Edit View Search Project Processor Expert Device Initialization Window Help
cfv1jm128_cdc.mcp
P&E Multilink/Cyclone Pro
Click to build project
Files Link Order Targets
Make

File	Code	Data
source	0	0
bsp	0	0
mcf51JM	0	0
usb_mcf51JM.c	0	0
app	0	0
cdc_serial.c	0	0
rtc.c	0	0
sci.c	0	0
classes	0	0
cdc	0	0
usb_host_cdc.c	0	0
usb_host_cdc_intf.c	0	0
hid	0	0
hub	0	0
msd	0	0
usb_classes.c	0	0
driver	0	0
khci.c	0	0
usbevent.c	0	0
usbmsgq.c	0	0
usbsem.c	0	0
host common	0	0
host_ch9.c	0	0
host_close.c	0	0
72 files	0	0

Figure A-9. Build a project

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

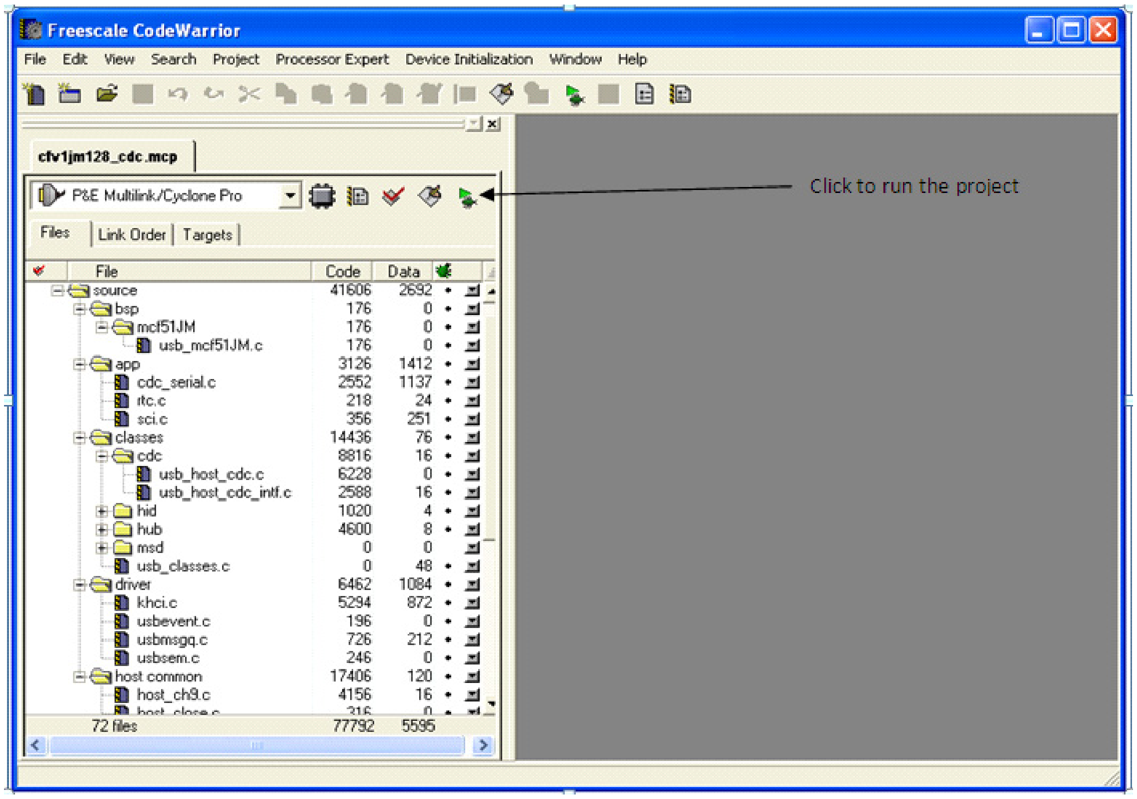
NOTE

Use the above mentioned steps to build the CFV2 projects also.

A.1.3 Running the application with CodeWarrior 6

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 application with DemoJM128 board using a P&E-micro debugger.

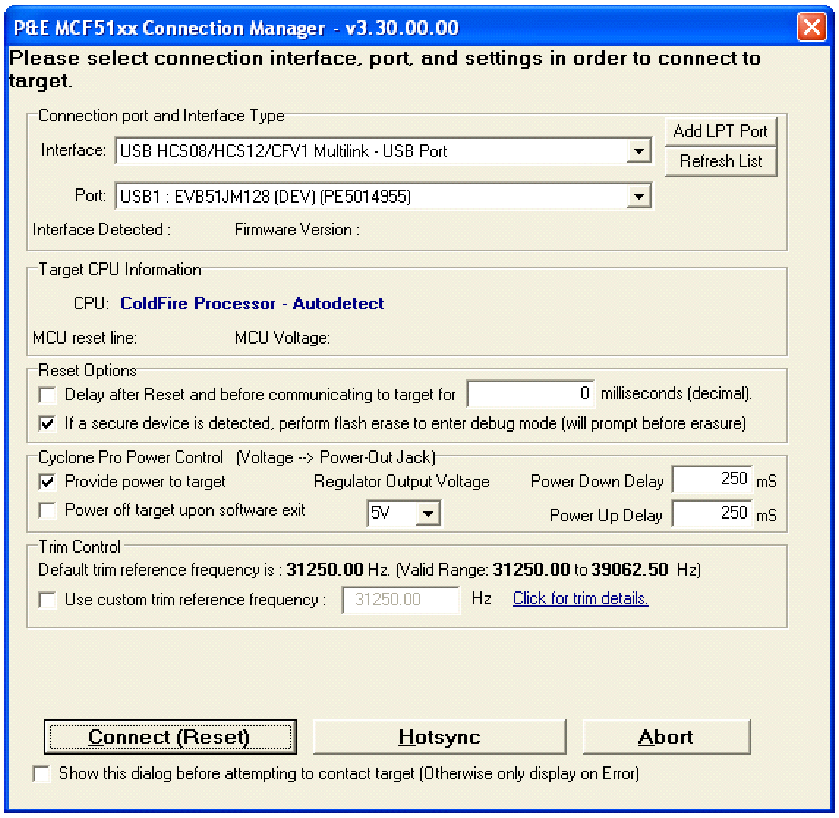
1. To run the application, click the button as shown in the following figure.


Freescale CodeWarrior
File Edit View Search Project Processor Expert Device Initialization Window Help
cfv1jm128_cdc.mcp
P&E Multilink/Cyclone Pro
Files
Link Order
Targets
Click to run the project

File	Code	Data
source	41606	2692
bsp	176	0
mcf51JM	176	0
usb_mcf51JM.c	176	0
app	3126	1412
cdc_serial.c	2552	1137
rtc.c	218	24
sci.c	356	251
classes	14436	76
cdc	8816	16
usb_host_cdc.c	6228	0
usb_host_cdc_intf.c	2588	16
hid	1020	4
hub	4600	8
msd	0	0
usb_classes.c	0	48
driver	6462	1084
khci.c	5294	872
usbevent.c	196	0
usbmsgq.c	726	212
usbsem.c	246	0
host common	17406	120
host_ch9.c	4156	16
host_close.c	316	0
72 files	77792	5595

Figure A-10. Running the application

2. Click the **Connect (Reset)** button to connect to hardware, as shown in the following figure.



P&E MCF51xx Connection Manager - v3.30.00.00

Please select connection interface, port, and settings in order to connect to target.

Connection port and Interface Type

Interface: USB HCS08/HCS12/CFV1 Multilink - USB Port

Port: USB1 : EVB51JM128 (DEV) (PE5014955)

Interface Detected :

Firmware Version :

Target CPU Information

CPU: ColdFire Processor - Autodetect

MCU reset line:

MCU Voltage:

Reset Options

☐ Delay after Reset and before communicating to target for 0 milliseconds (decimal).

☑ If a secure device is detected, perform flash erase to enter debug mode (will prompt before erasure)

Cyclone Pro Power Control (Voltage --> Power-Out Jack)

☑ Provide power to target

☐ Power off target upon software exit

Regulator Output Voltage 5V

Power Down Delay 250 mS

Power Up Delay 250 mS

Trim Control

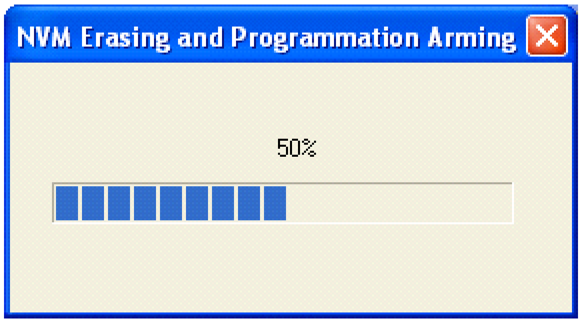
Default trim reference frequency is : **31250.00** Hz. (Valid Range: **31250.00** to **39062.50** Hz)

☐ Use custom trim reference frequency : 31250.00 Hz Click for trim details.

☐ Show this dialog before attempting to contact target (Otherwise only display on Error)

Figure A-11. Connection manager

3. The pop-up in the following figure shows the progress while erasing and programming the built image to JM128 flash.

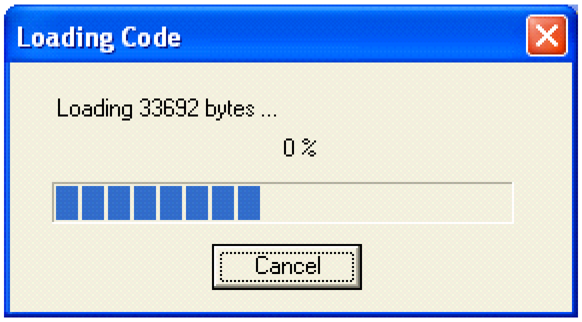


NVM Erasing and Programmation Arming

50%

Figure A-12. Erasing and programming window

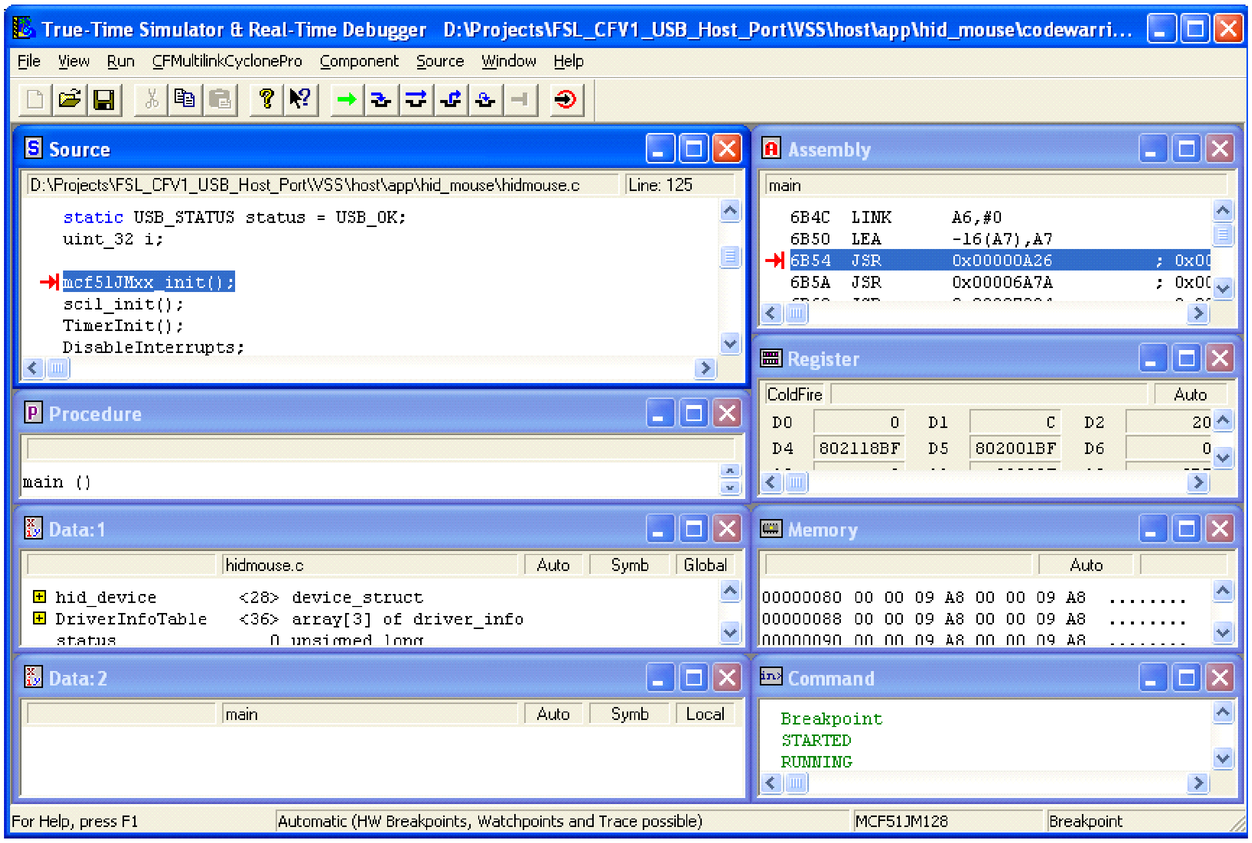
4. The pop-up in the following figure shows the progress while loading the built image to JM128 flash.



Loading 33692 bytes ...
0 %

Figure A-13. Loading window

5. After the built image is loaded in the flash, the debugger window shown in the following figure appears. Click on the green arrow as shown to run the programmed image.


True-Time Simulator & Real-Time Debugger D:\Projects\FSL_CFV1_USB_Host_Port\VSS\host\app\hid_mouse\codewarri...
File View Run CFMultilinkCyclonePro Component Source Window Help
Source
Assembly
Register
Procedure
Memory
Data:1
Data:2
Command

D:\Projects\FSL_CFV1_USB_Host_Port\VSS\host\app\hid_mouse\hidmouse.c Line: 125

```
    static USB_STATUS status = USB_OK;
    uint_32 i;

    mcf51JMxx_init();
    scil_init();
    TimerInit();
    DisableInterrupts;
```

main

```
  6B4C   LINK      A6,#0
  6B50   LEA       -16(A7),A7
  6B54   JSR       0x00000A26          ; 0x00
  6B5A   JSR       0x00006A7A          ; 0x00
```

ColdFire					Auto
D0	0	D1	C	D2	20
D4	802118BF	D5	802001BF	D6	0

main ()

hidmouse.c Auto Symb Global

```
 hid_device          <28>  device_struct
 DriverInfoTable     <36>  array[3] of driver_info
 status                0  unsigned long
```

Auto

```
00000080  00 00 09 A8 00 00 09 A8   ........
00000088  00 00 09 A8 00 00 09 A8   ........
00000090  00 00 09 A8 00 00 09 A8   ........
```

main Auto Symb Local

```
Breakpoint
STARTED
RUNNING
```

For Help, press F1 | Automatic (HW Breakpoints, Watchpoints and Trace possible) | MCF51JM128 | Breakpoint

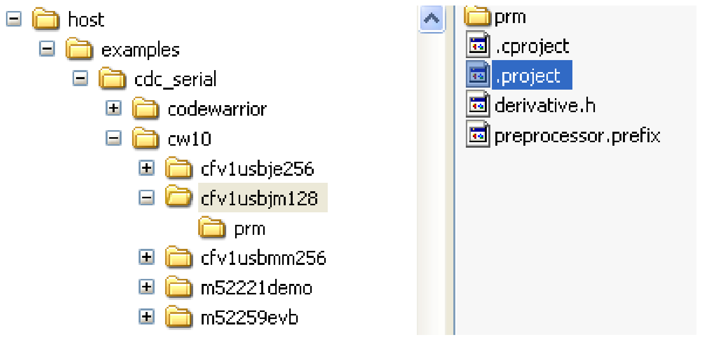
A.1.4 Building and running the application with CodeWarrior 10

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

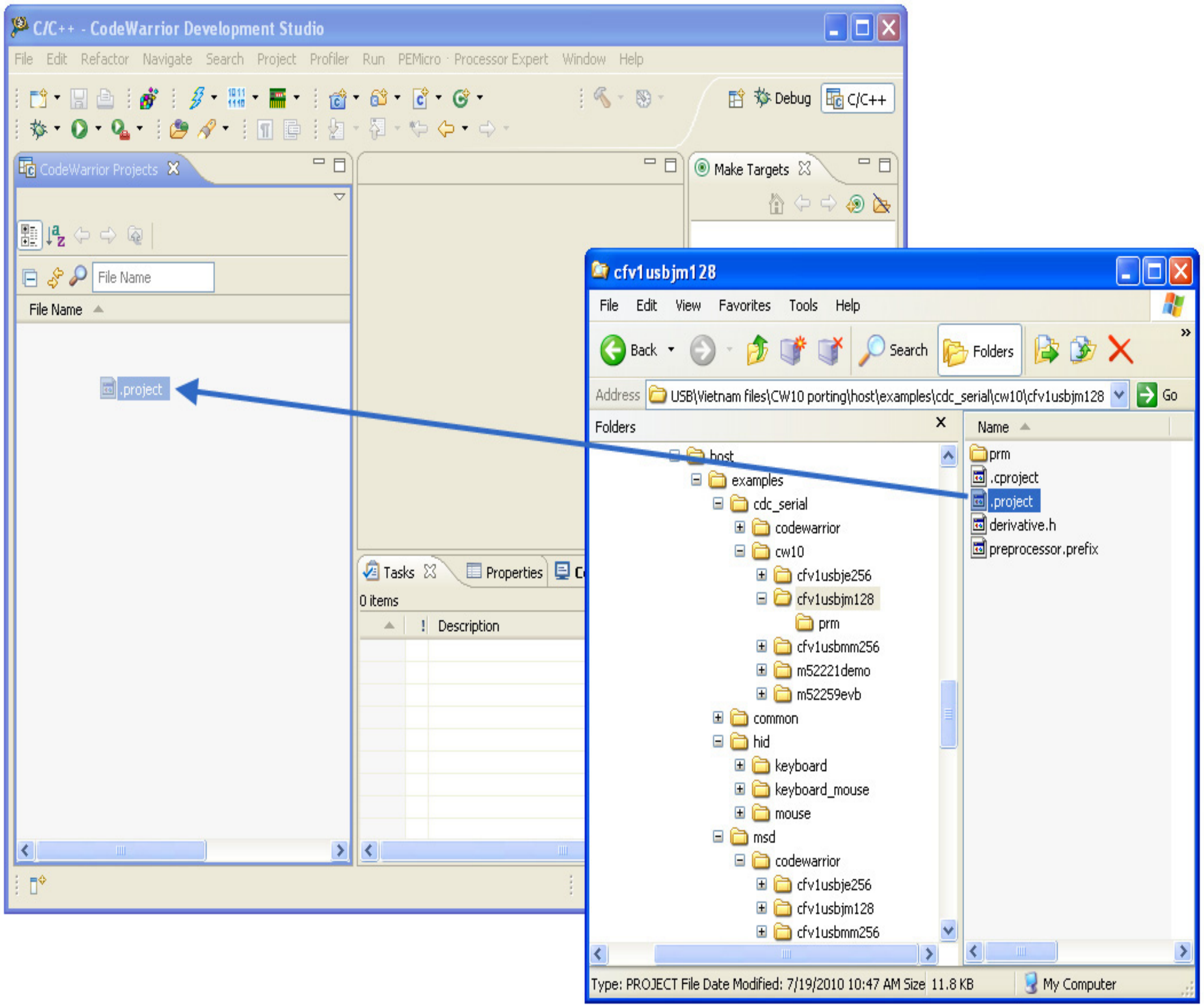
Before starting the process of building the project, make sure CodeWarrior 10 MCU is installed on your computer.

To build the S08/CFV1/CFV2 project:

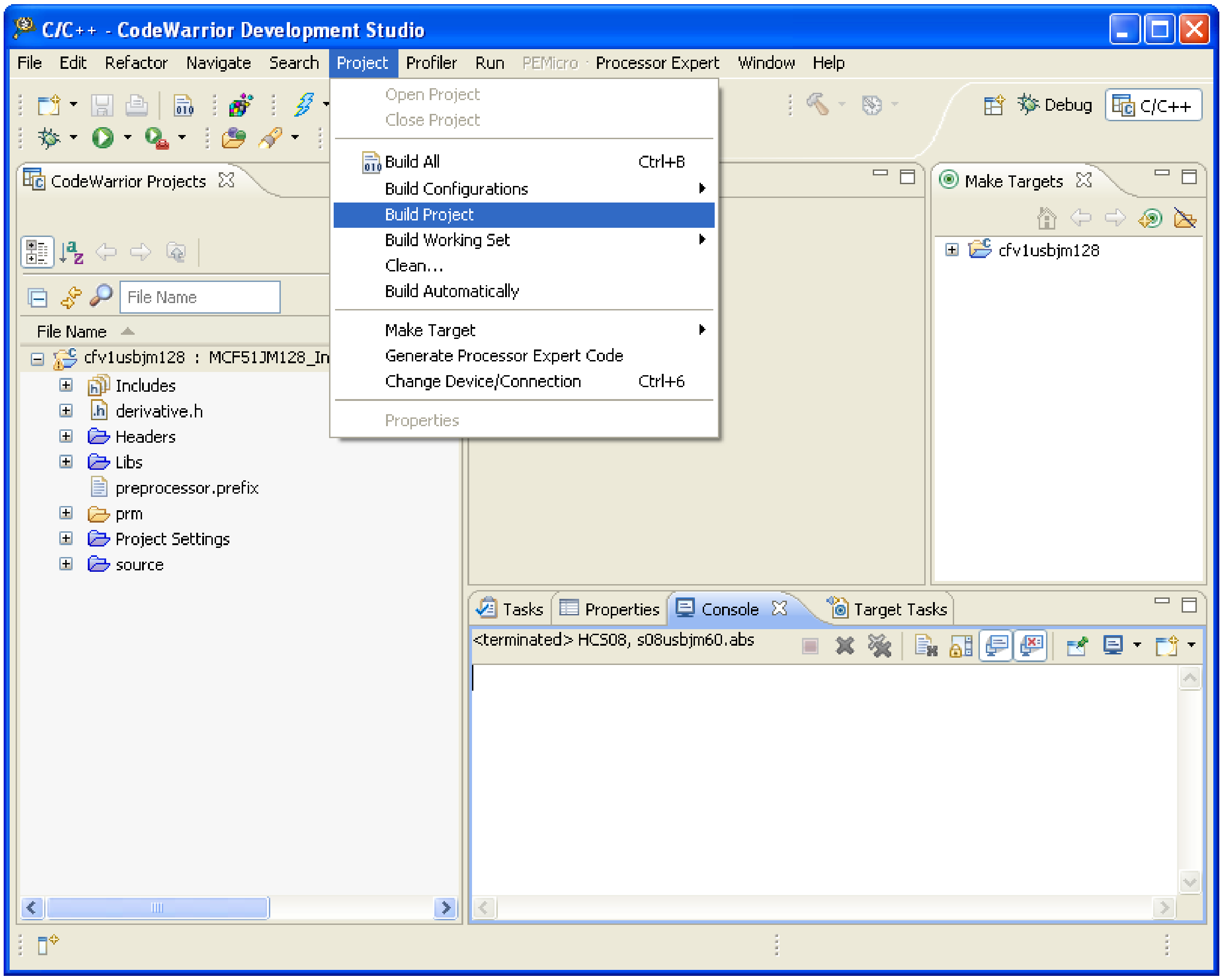
1. Navigate to the project folder (cfv1usbjm128) and locate the CodeWarrior10 project file (.project).



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.

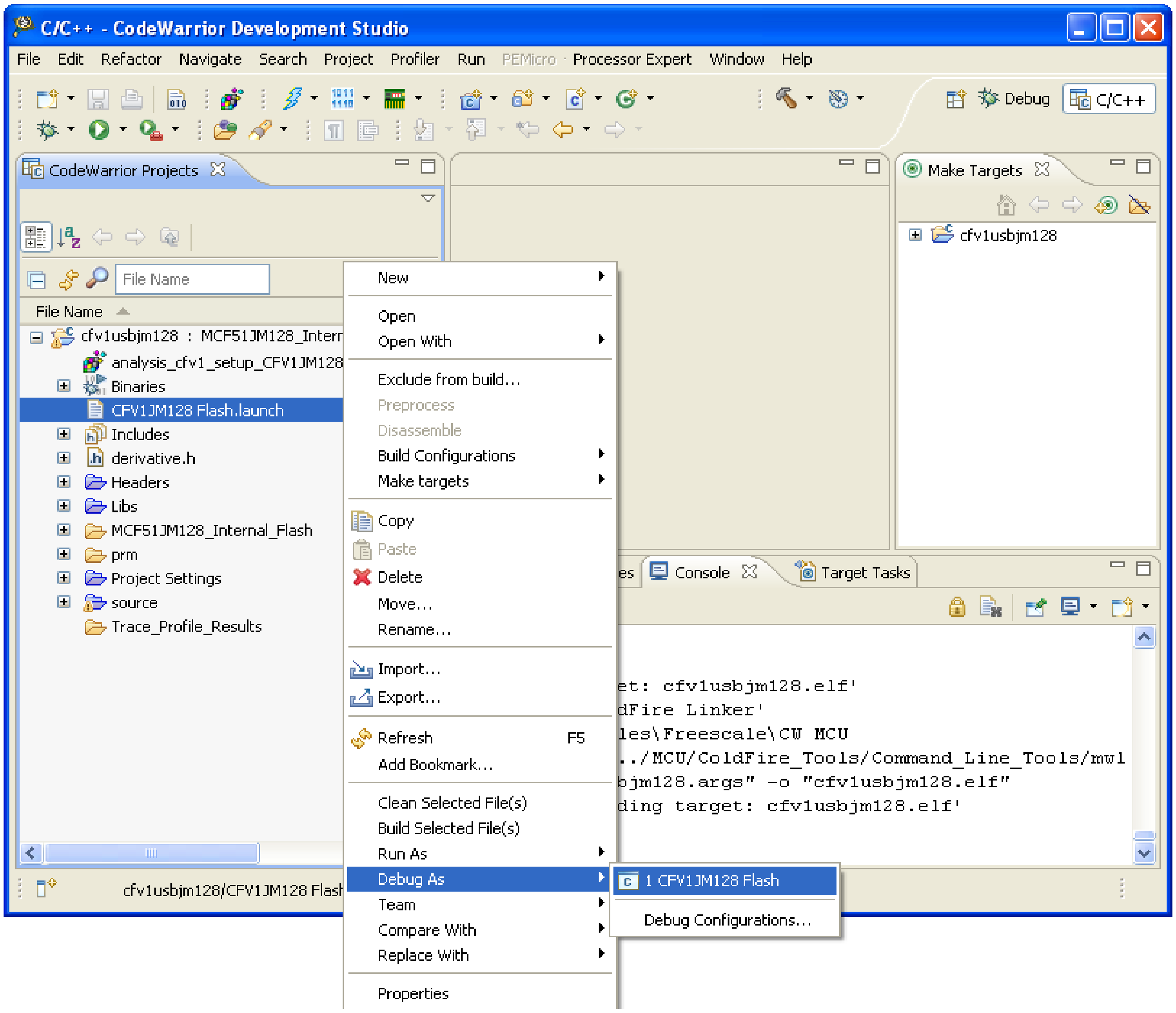


<terminated> HCS08, s08usbjm60.abs

NOTE

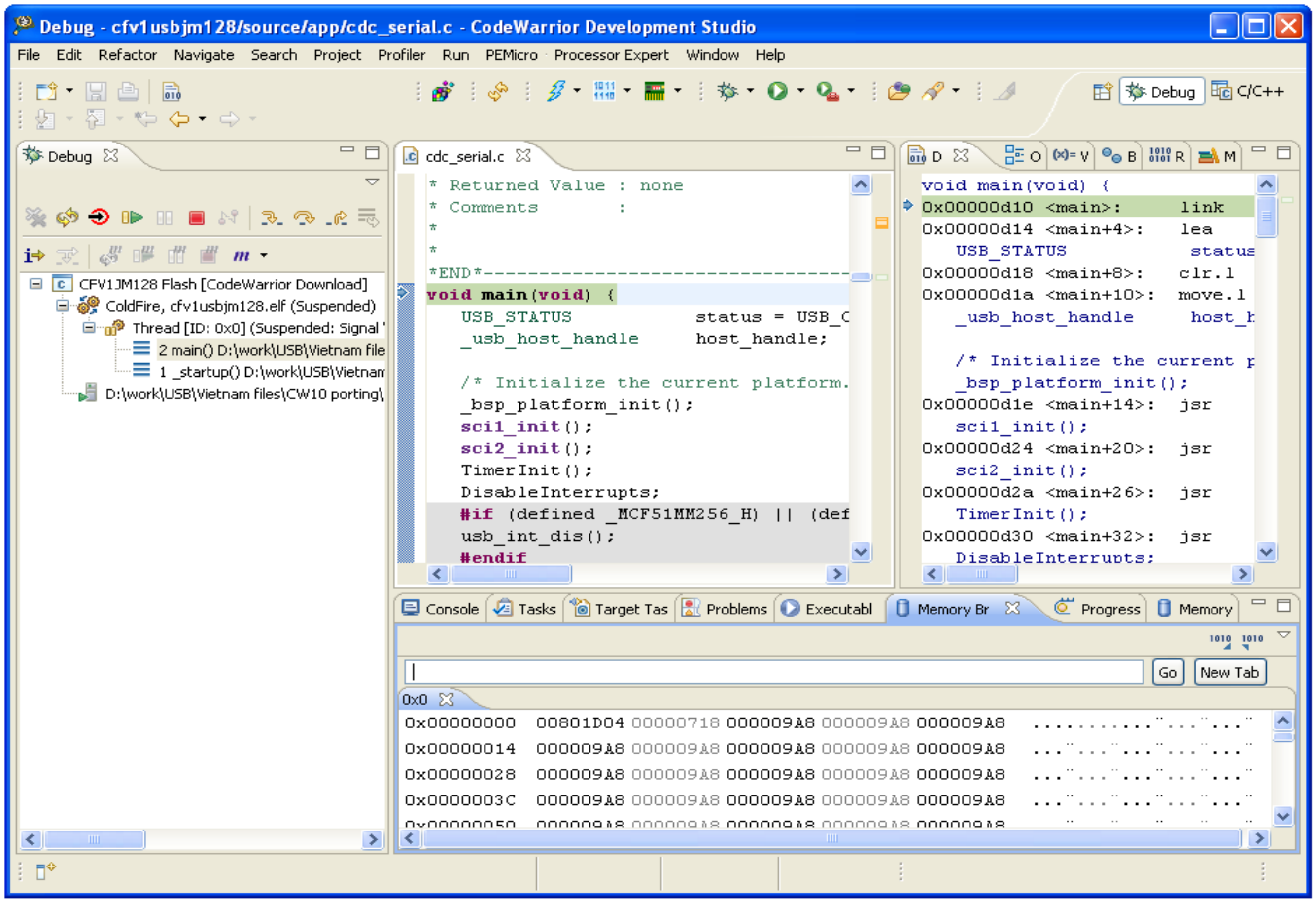
You must follow the above procedure to build CFV2 projects also.

4. To run the application, first locate the CFV1JM128 Flash.launch configuration in the current project space. Right-click it and choose Debug As > 1 CFV1JM128 Flash as in the window below.



```
et: cfv1usbjm128.elf'
dFire Linker'
les\Freescale\CW MCU
../MCU/ColdFire_Tools/Command_Line_Tools/mwl
bjm128.args" -o "cfv1usbjm128.elf"
ding target: cfv1usbjm128.elf'
```

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


Debug - cfv1usbjm128/source/app/cdc_serial.c - CodeWarrior Development Studio
File Edit Refactor Navigate Search Project Profiler Run PEMicro Processor Expert Window Help
Debug
C/C++
Debug
CFV1JM128 Flash [CodeWarrior Download]
ColdFire, cfv1usbjm128.elf (Suspended)
Thread [ID: 0x0] (Suspended: Signal '
2 main() D:\work\USB\Vietnam file
1 _startup() D:\work\USB\Vietnam
D:\work\USB\Vietnam files\CW10 porting\
cdc_serial.c
D
O
V
B
R
M
Console
Tasks
Target Tas
Problems
Executabl
Memory Br
Progress
Memory
Go
New Tab
0x0

```
* Returned Value : none
* Comments       :
*
*
*END*------------------------------------
void main(void) {
   USB_STATUS           status = USB_C
   _usb_host_handle     host_handle;

   /* Initialize the current platform.
   _bsp_platform_init();
   sci1_init();
   sci2_init();
   TimerInit();
   DisableInterrupts;
   #if (defined _MCF51MM256_H) || (def
   usb_int_dis();
   #endif
```

```
void main(void) {
0x00000d10 <main>:      link
0x00000d14 <main+4>:    lea
   USB_STATUS            status
0x00000d18 <main+8>:    clr.l
0x00000d1a <main+10>:   move.l
   _usb_host_handle      host_h

   /* Initialize the current p
   _bsp_platform_init();
0x00000d1e <main+14>:   jsr
   sci1_init();
0x00000d24 <main+20>:   jsr
   sci2_init();
0x00000d2a <main+26>:   jsr
   TimerInit();
0x00000d30 <main+32>:   jsr
   DisableInterrupts;
```

```
0x00000000   00801D04 00000718 000009A8 000009A8 000009A8
0x00000014   000009A8 000009A8 000009A8 000009A8 000009A8
0x00000028   000009A8 000009A8 000009A8 000009A8 000009A8
0x0000003C   000009A8 000009A8 000009A8 000009A8 000009A8
0x00000050   000009A8 000009A8 000009A8 000009A8 000009A8
```

A.2 Set up HyperTerminal to get log

To ensure that applications run correctly, the HyperTerminal is used on your computer to get events from the devices that connect to the CFV1 and CFV2. These steps are used to configure HyperTerminal:

1. Open HyperTerminal applications as shown in the following figure.

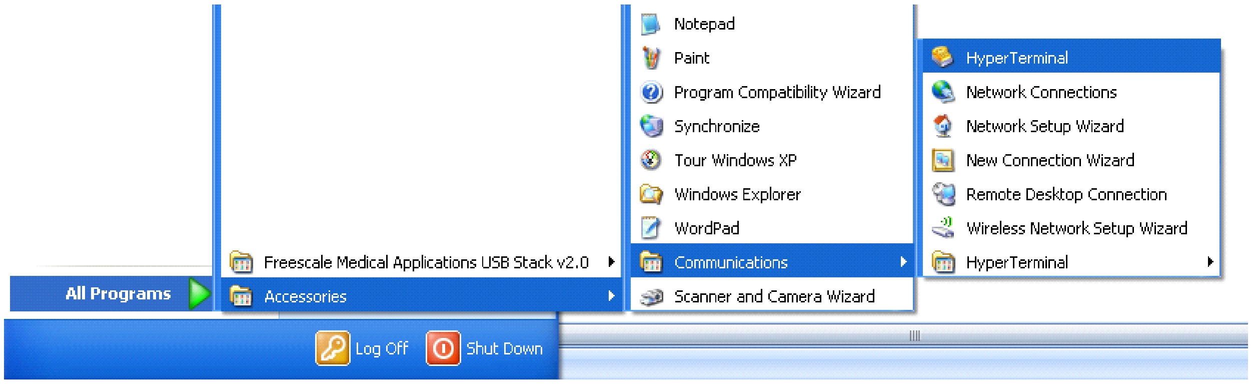
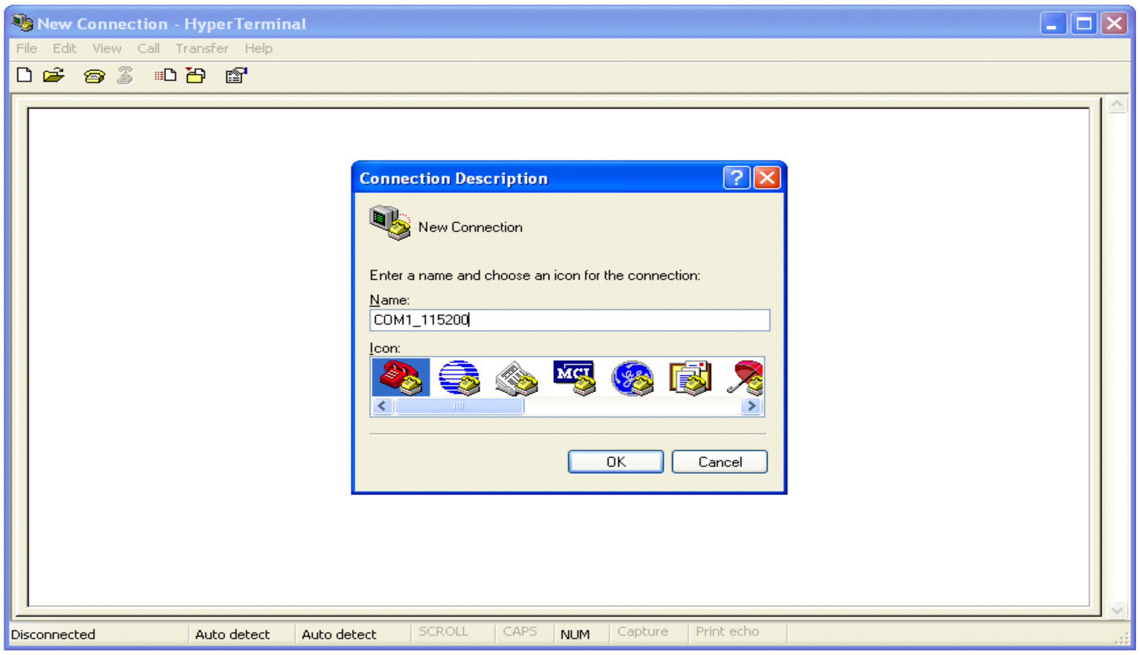


Figure A-14. Launch HyperTerminal application

2. The HyperTerminal opens as shown in the following figure. Enter the name of the connection and click on the **OK** button.



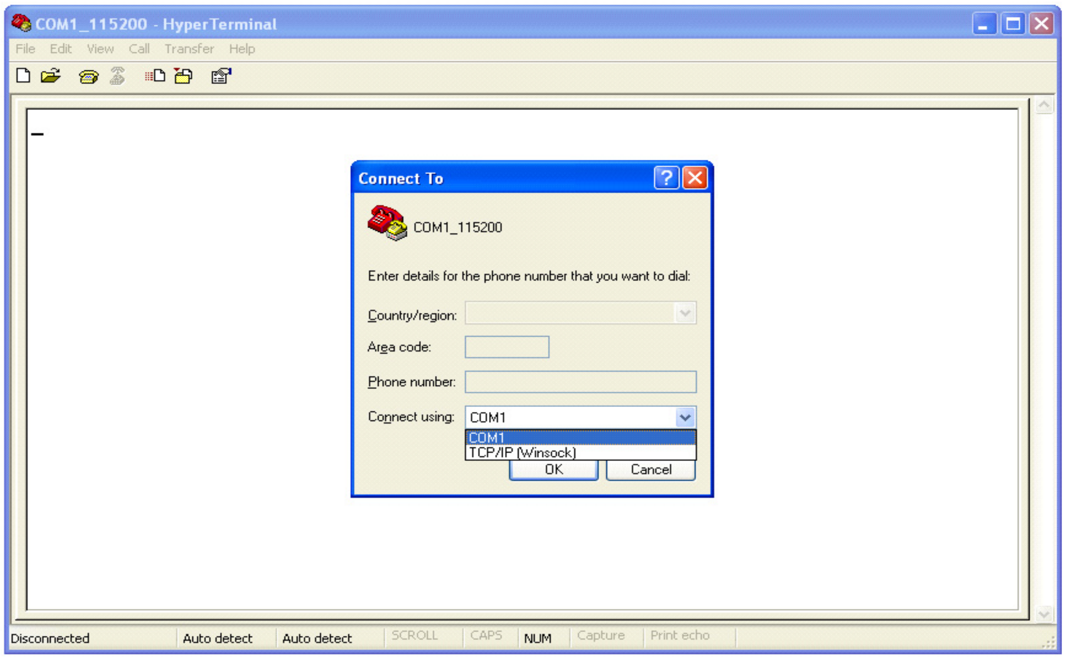
Enter a name and choose an icon for the connection:

Name:

COM1_115200

Figure A-15. HyperTerminal GUI

3. The window shown in the following figure appears. Select the COM port identical to the one that shows up on the device manager.



Connect To

COM1_115200

Enter details for the phone number that you want to dial:

Country/region:

Area code:

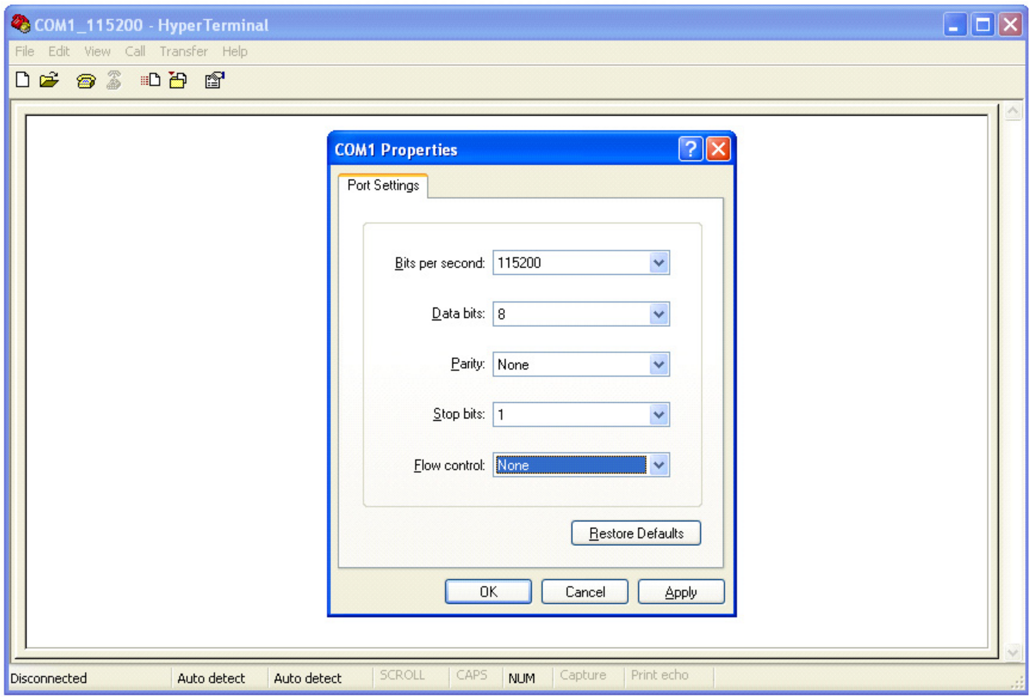
Phone number:

Connect using: COM1

COM1
TCP/IP (Winsock)

Figure A-16. Connect using COM1

4. Configure the virtual COM port baud rate and other properties as shown in the following figure.



COM1 Properties

Port Settings

Bits per second: 115200

Data bits: 8

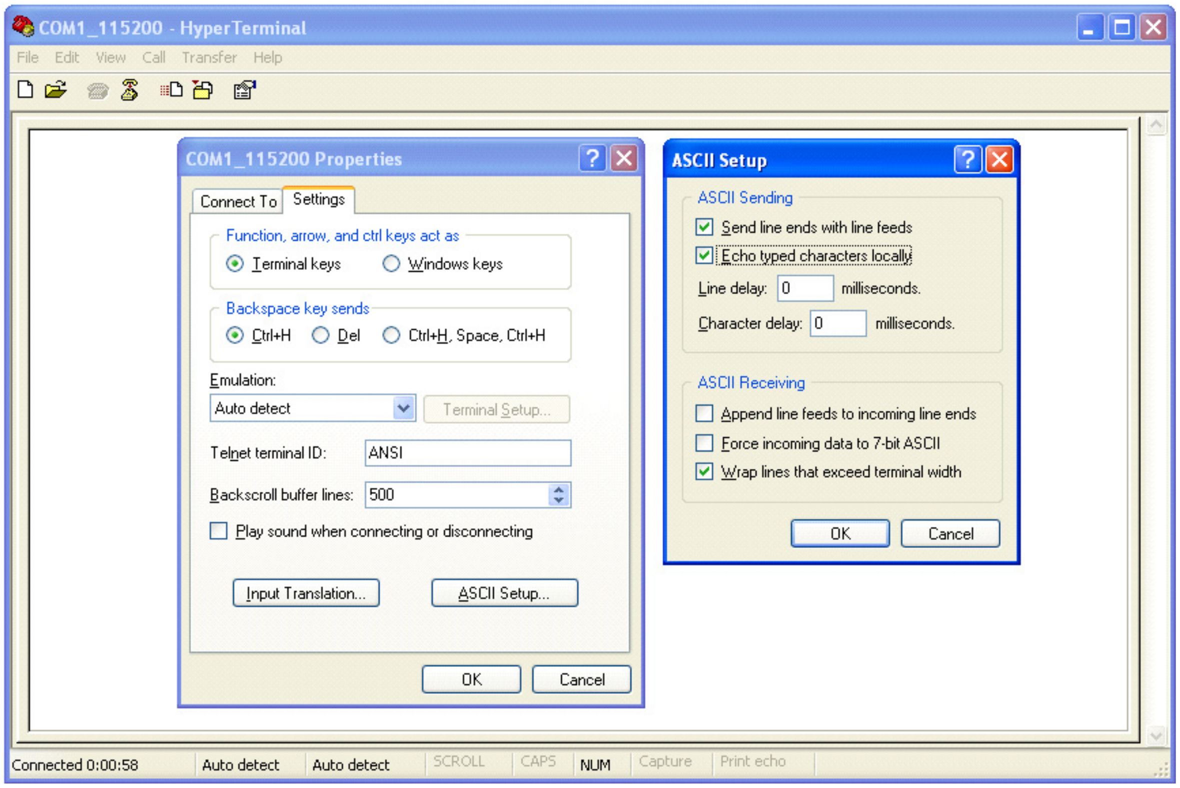
Parity: None

Stop bits: 1

Flow control: None

Figure A-17. COM1 properties

5. Configure the HyperTerminal as shown in the following figure. Click on the **OK** button to submit changes.



Function, arrow, and ctrl keys act as: [✓ Terminal keys] Windows keys

Backspace key sends: [✓ Ctrl+H] Del Ctrl+H, Space, Ctrl+H

Emulation: Auto detect

Telnet terminal ID: ANSI

Backscroll buffer lines: 500

☐ Play sound when connecting or disconnecting

ASCII Sending

☑ Send line ends with line feeds

☑ Echo typed characters locally

Line delay: 0 milliseconds.

Character delay: 0 milliseconds.

ASCII Receiving

☐ Append line feeds to incoming line ends

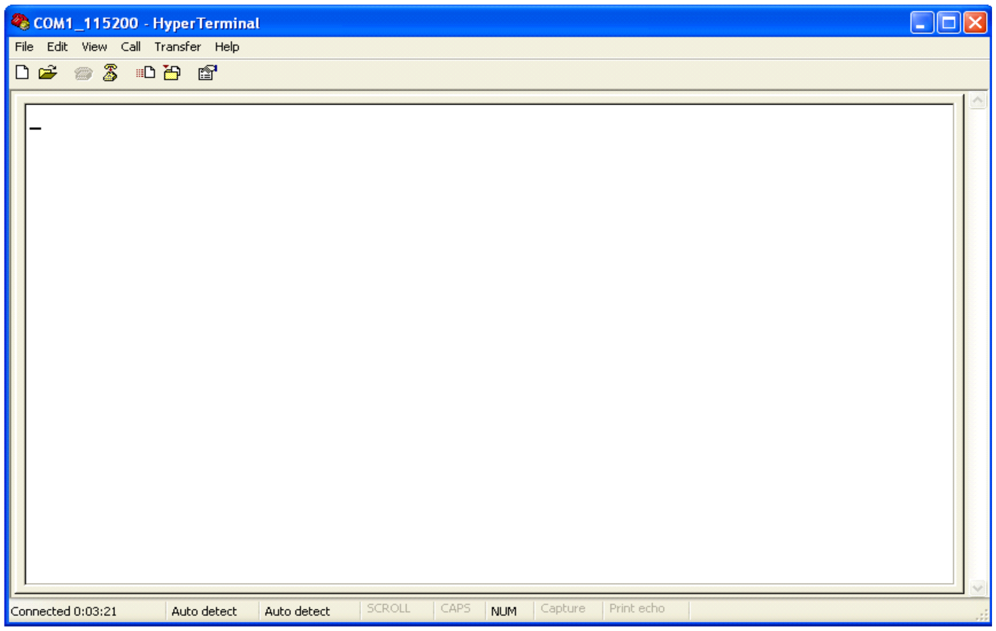
☐ Force incoming data to 7-bit ASCII

☑ Wrap lines that exceed terminal width

Connected 0:00:58 Auto detect Auto detect SCROLL CAPS NUM Capture Print echo

Figure A-18. Configure COM1_115200 — HyperTerminal

6. The HyperTerminal is now configured.


COM1_115200 - HyperTerminal
File Edit View Call Transfer Help

Connected 0:03:21 Auto detect Auto detect SCROLL CAPS NUM Capture Print echo

Figure A-19. COM1_115200 is configured

A.3 Uninstall Freescale USB Stack Software

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

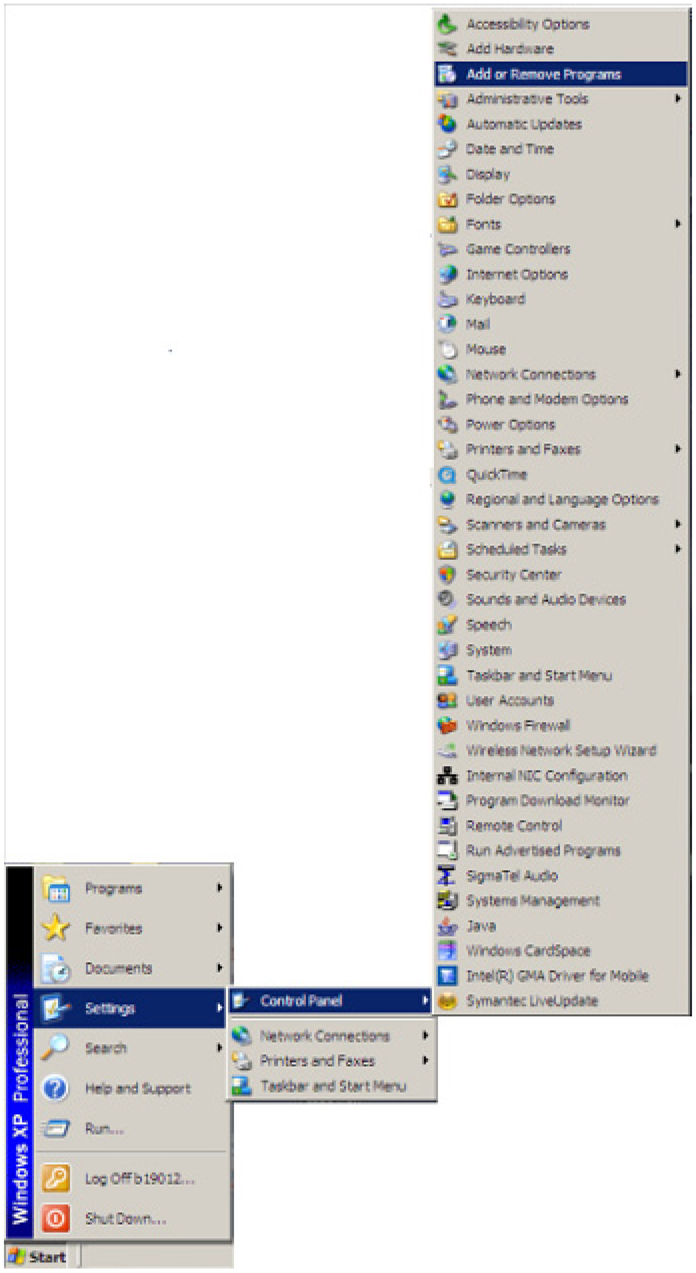
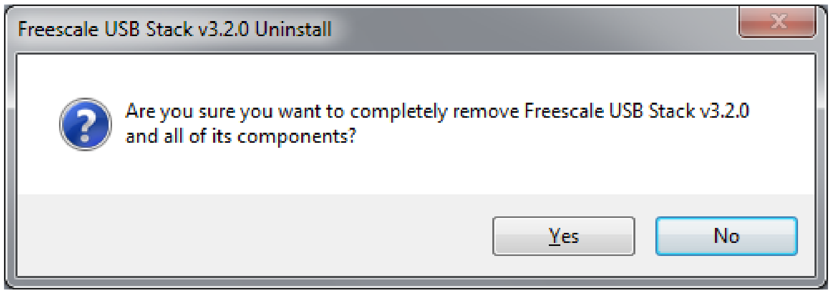


Figure A-20. Launch "Add or Remove Programs" from Control Panel

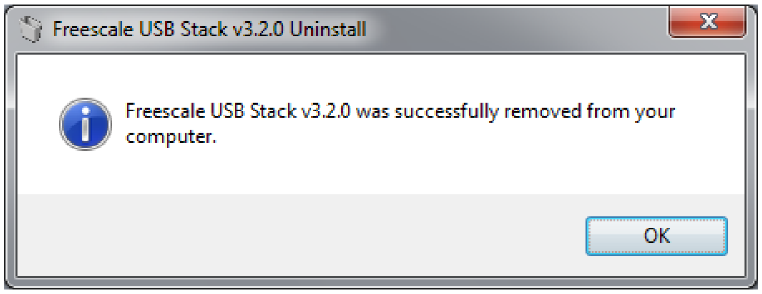
2. In the Windows Control Panel “Add/Romove Programs” Toolselect **Freescale USB Stack** and click on the **Change/Remove** button.
3. The uninstall confirmation message appears. Click on **Yes** button to uninstall.


Freescale USB Stack v3.2.0 Uninstall

Are you sure you want to completely remove Freescale USB Stack v3.2.0 and all of its components?

Figure A-21. Freescale USB Stack uninstall confirmation message

4. A message box appears. Click on the **OK** button to complete the uninstall operation.


Freescale USB Stack v3.2.0 Uninstall

Freescale USB Stack v3.2.0 was successfully removed from your computer.

Figure A-22. Freescale USB Stack uninstall completion message

Appendix B Human Interface Device (HID) Demo

B.1 Setting up the demo

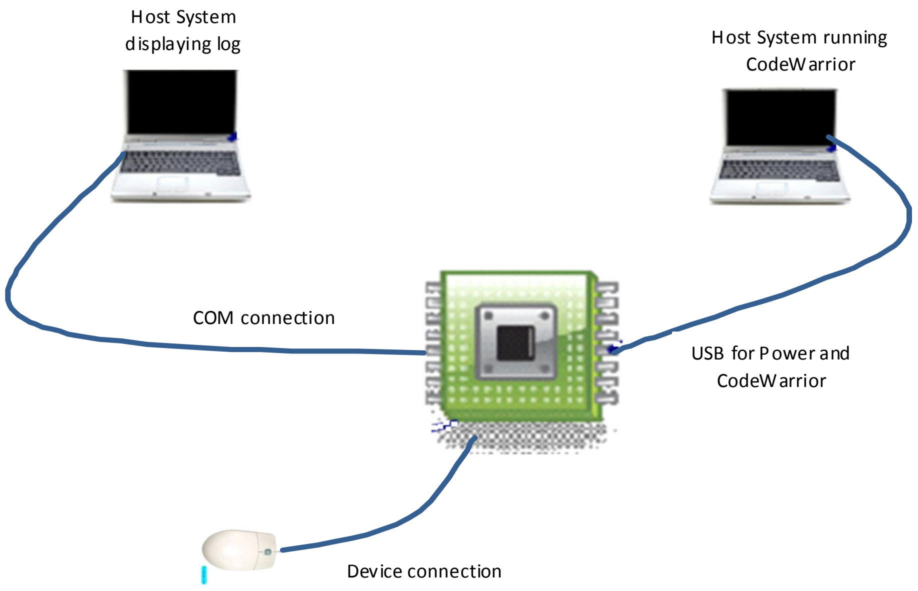


Figure B-1. HID demo setup

The preceding figure shows the HID demo setup. The DemoJM is used as the USB host. DemoJM is connected to the first personal computer using USB cables. This computer is used to supply power to the board and is used to program the image to the flash. DemoJM is also connected to the second personal computer via a COM port. This computer is used to log events happening in the USB host. The device (mouse or keyboard) is connected to DemoJM. Although the proceeding figure shows two computers, the connection can also be achieved using only one computer.

B.2 Running the demo

The HID project is located in **\Freescale USB Stack\Source\USB Host\HID Class Demo Apps**.

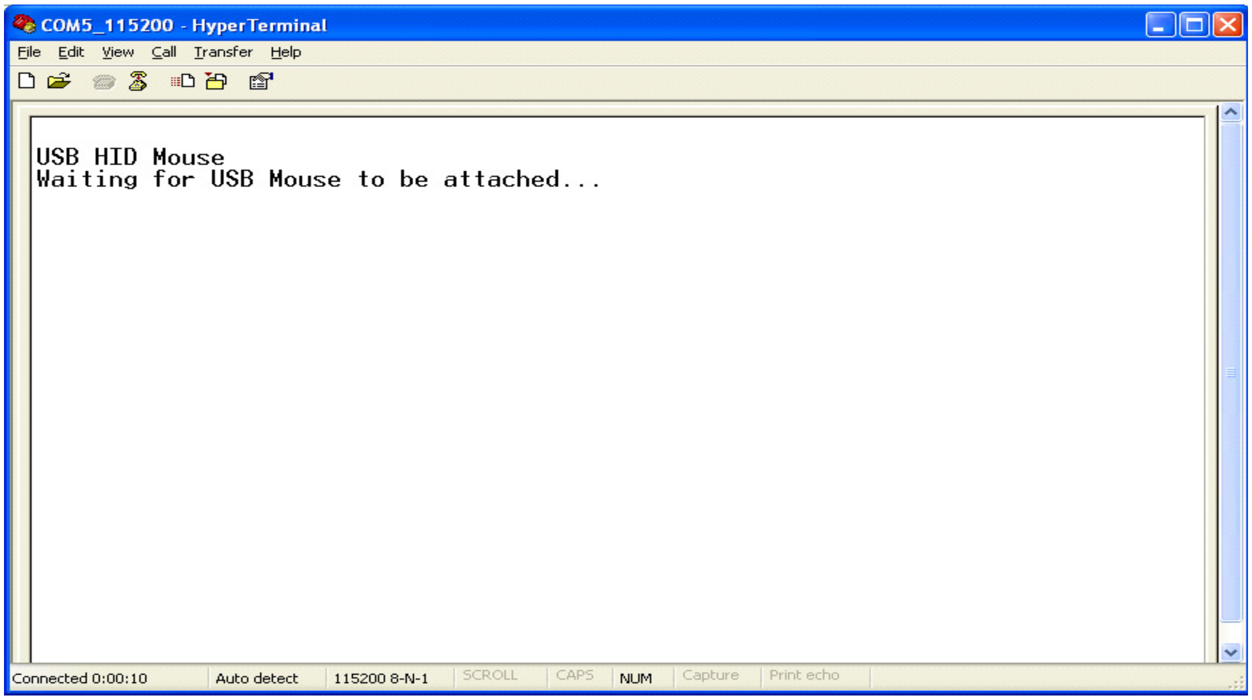
There are three applications of HID classes:

- Mouse
- Keyboard
- Keyboard and mouse

B.2.1 Mouse demo

Perform the following steps to run the mouse demo:

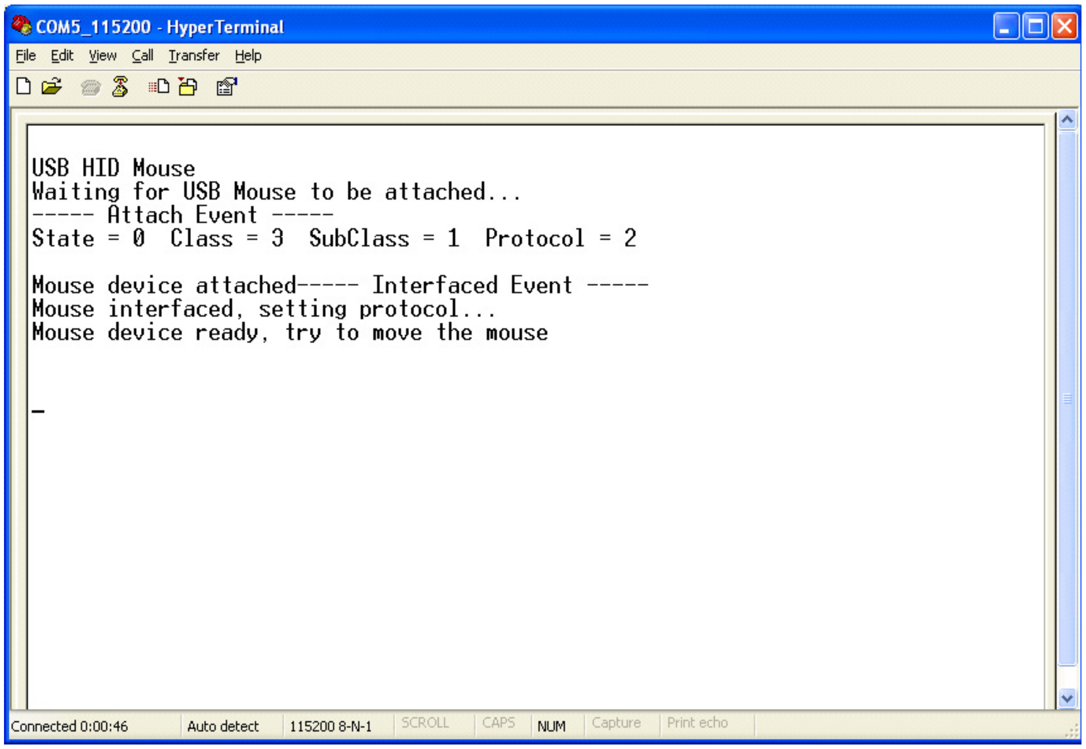
1. Open and load the image of mouse applications to the board.
2. After the image has been loaded successfully, HyperTerminal appears as shown in the following figure.


COM5_115200 - HyperTerminal
File Edit View Call Transfer Help

```
USB HID Mouse
Waiting for USB Mouse to be attached...
```

Figure B-2. USB Host waiting for mouse attachment event

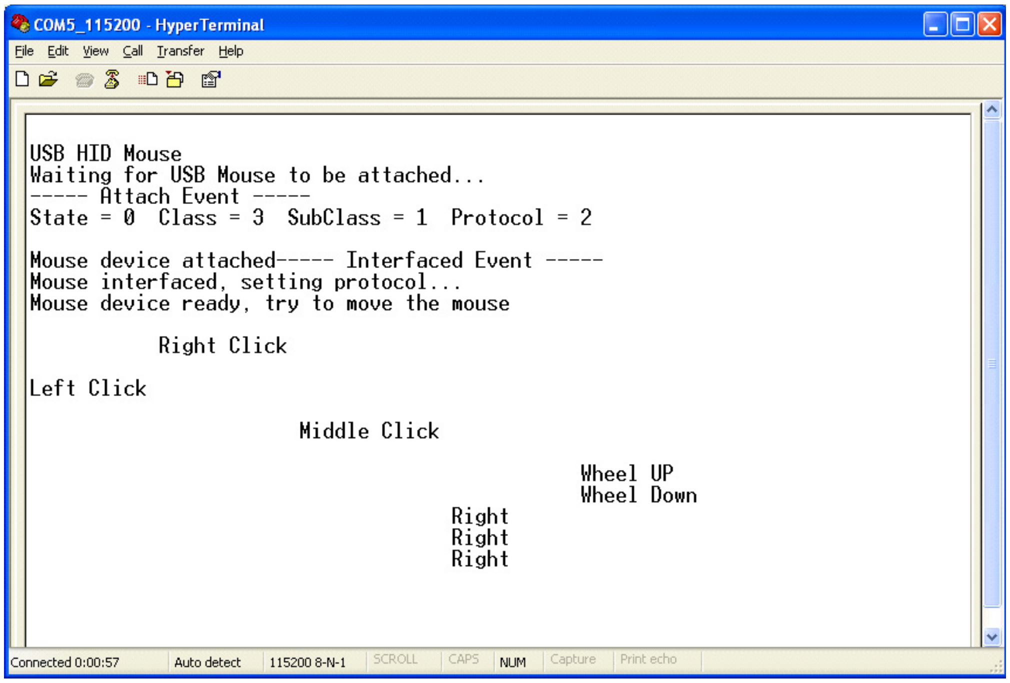
3. Plug the mouse into the board. The Hyperterminal screen appears as shown in the following figure.



```
USB HID Mouse
Waiting for USB Mouse to be attached...
----- Attach Event -----
State = 0  Class = 3  SubClass = 1  Protocol = 2

Mouse device attached----- Interfaced Event -----
Mouse interfaced, setting protocol...
Mouse device ready, try to move the mouse



_
```

Figure B-3. Mouse attached

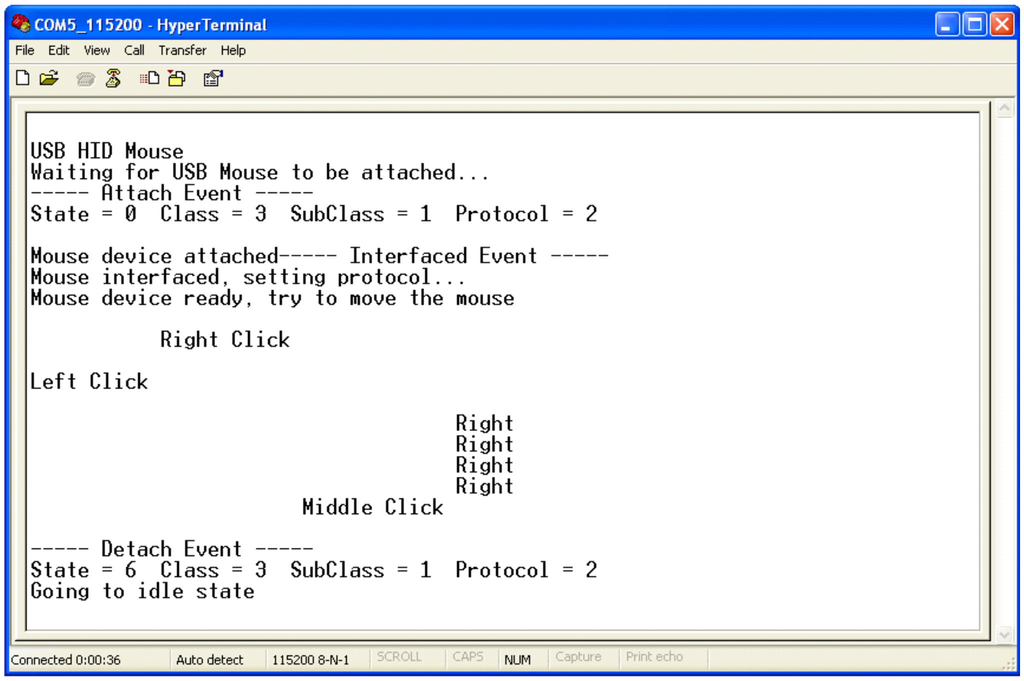
4. When events are implemented (click right mouse, click left mouse, and so on), they are registered as shown in the following figure.



```
USB HID Mouse
Waiting for USB Mouse to be attached...
----- Attach Event -----
State = 0  Class = 3  SubClass = 1  Protocol = 2

Mouse device attached----- Interfaced Event -----
Mouse interfaced, setting protocol...
Mouse device ready, try to move the mouse

           Right Click

Left Click

                       Middle Click

                                               Wheel UP
                                               Wheel Down
                                    Right
                                    Right
                                    Right
```

Figure B-4. Events from mouse

5. Unplug the mouse from the board. HyperTerminal displays a message as shown in the following figure.



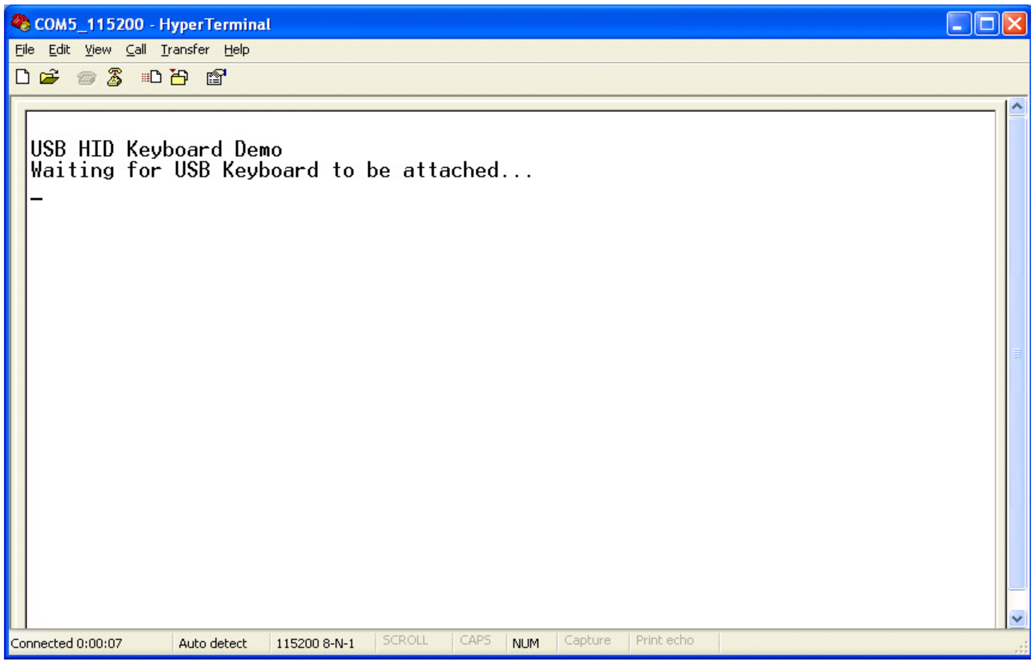
```
USB HID Mouse
Waiting for USB Mouse to be attached...
----- Attach Event -----
State = 0  Class = 3  SubClass = 1  Protocol = 2

Mouse device attached----- Interfaced Event -----
Mouse interfaced, setting protocol...
Mouse device ready, try to move the mouse

           Right Click

Left Click

                                    Right
                                    Right
                                    Right
                                    Right
                       Middle Click

----- Detach Event -----
State = 6  Class = 3  SubClass = 1  Protocol = 2
Going to idle state
```

Figure B-5. Mouse detached

B.2.2 Keyboard demo

Perform the following steps to run the keyboard demo:

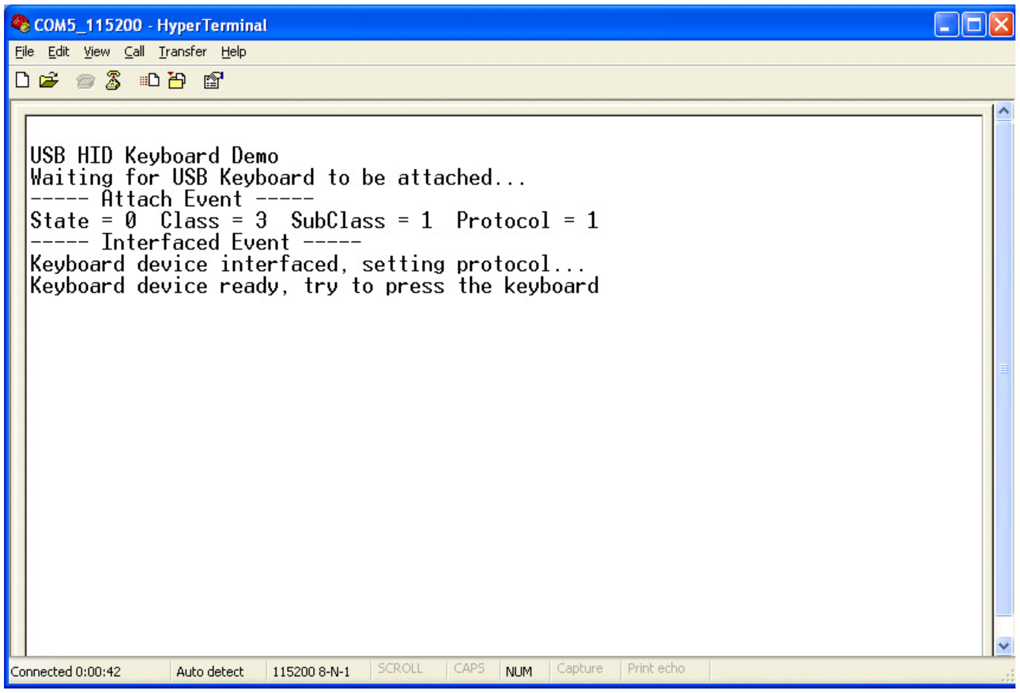
1. Open and load the image to the board.
6. Run the demo. First, the USB host waits for the device attachment event. The HyperTerminal displays a message as shown in the following figure.



```
USB HID Keyboard Demo
Waiting for USB Keyboard to be attached...
_
```

Figure B-6. USB host waiting for keyboard attachment event

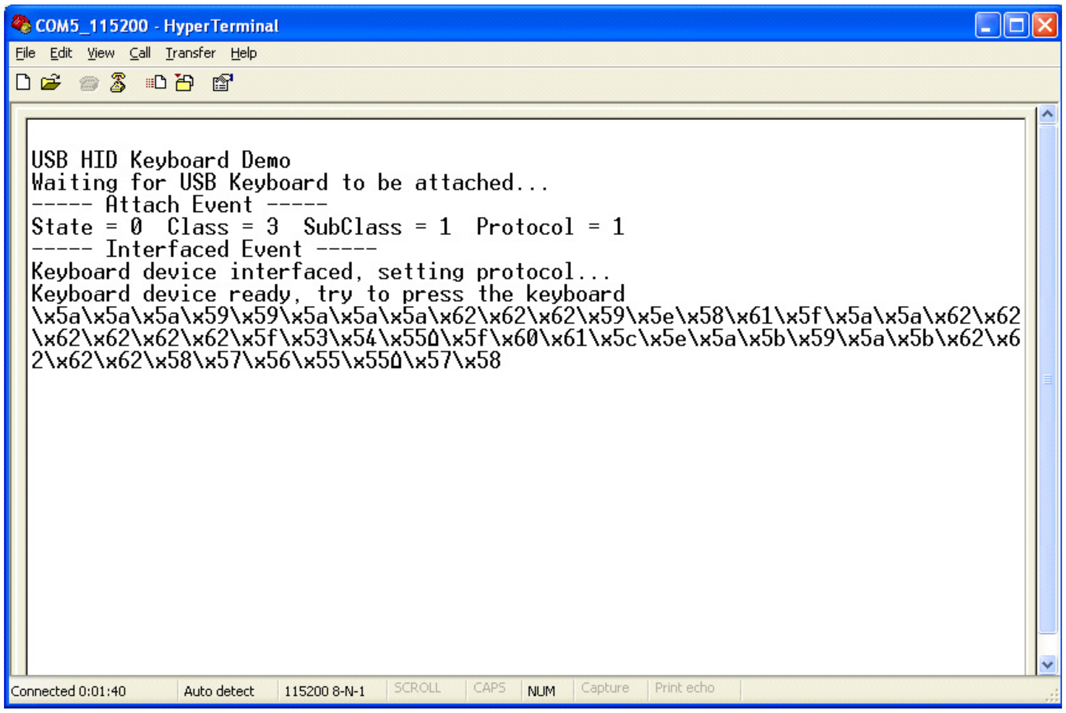
2. Plug in keyboard. HyperTerminal shows a message as in the following figure.



```
USB HID Keyboard Demo
Waiting for USB Keyboard to be attached...
----- Attach Event -----
State = 0  Class = 3  SubClass = 1  Protocol = 1
----- Interfaced Event -----
Keyboard device interfaced, setting protocol...
Keyboard device ready, try to press the keyboard
```

Figure B-7. Keyboard attached

3. Type some characters. The hexadecimal format of these characters will be displayed in HyperTerminal as shown in the following figure.



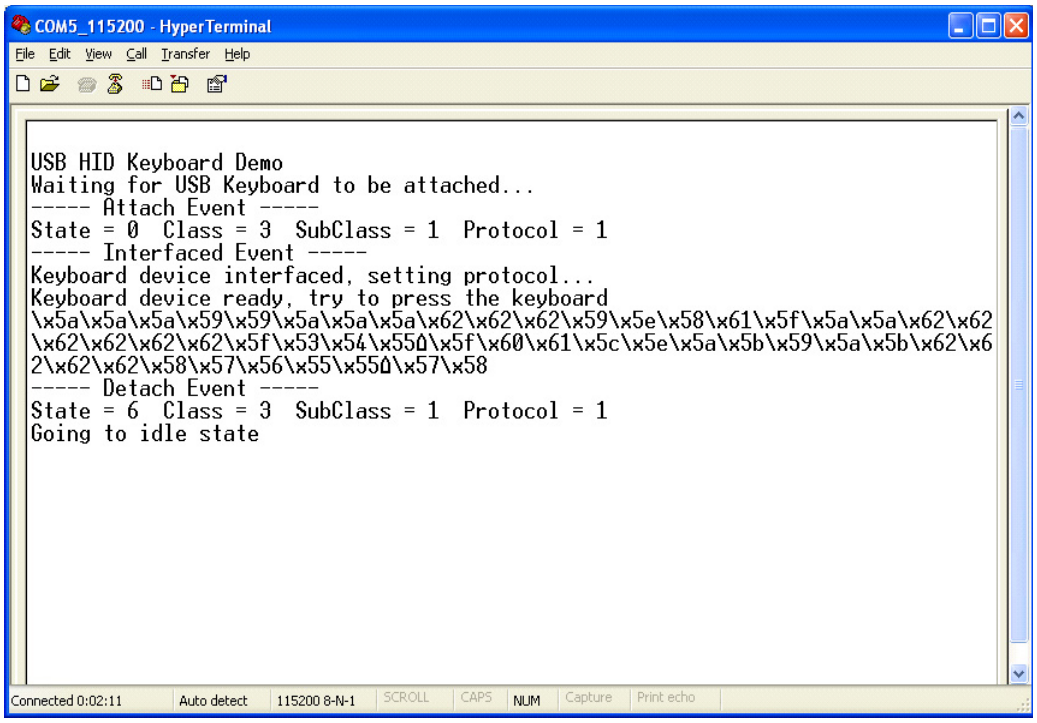
```
USB HID Keyboard Demo
Waiting for USB Keyboard to be attached...
----- Attach Event -----
State = 0  Class = 3  SubClass = 1  Protocol = 1
----- Interfaced Event -----
Keyboard device interfaced, setting protocol...
Keyboard device ready, try to press the keyboard
\x5a\x5a\x5a\x59\x59\x5a\x5a\x5a\x62\x62\x62\x59\x5e\x58\x61\x5f\x5a\x5a\x62\x62
\x62\x62\x62\x62\x5f\x53\x54\x55Ω\x5f\x60\x61\x5c\x5e\x5a\x5b\x59\x5a\x5b\x62\x6
2\x62\x62\x58\x57\x56\x55\x55Ω\x57\x58
```

Figure B-8. Characters from keyboard

NOTE

The HyperTerminal shows only the ASCII code, in hexadecimal, of each character.

4. Unplug the keyboard. The HyperTerminal shows a message as shown in the following figure.



```
USB HID Keyboard Demo
Waiting for USB Keyboard to be attached...
----- Attach Event -----
State = 0  Class = 3  SubClass = 1  Protocol = 1
----- Interfaced Event -----
Keyboard device interfaced, setting protocol...
Keyboard device ready, try to press the keyboard
\x5a\x5a\x5a\x59\x59\x5a\x5a\x5a\x62\x62\x62\x59\x5e\x58\x61\x5f\x5a\x5a\x62\x62
\x62\x62\x62\x62\x5f\x53\x54\x55Δ\x5f\x60\x61\x5c\x5e\x5a\x5b\x59\x5a\x5b\x62\x6
2\x62\x62\x58\x57\x56\x55\x55Δ\x57\x58
----- Detach Event -----
State = 6  Class = 3  SubClass = 1  Protocol = 1
Going to idle state
```

Figure B-9. Keyboard detached

B.2.3 Mouse and keyboard demo

This application combines the two above. It supplies a convenient choice for users by allowing them to alternate which HID device (the mouse or the keyboard) they want to work with.

Appendix C Virtual Communication (COM) Demo

The USB-to-serial demo implements the Abstract Control Model (ACM) subclass of the USB CDC class, enabling the serial port applications on the host PC to transmit and receive serial data over the USB port.

C.1 Setting up the demo

Set up the system as described in Appendix B "Human Interface Device (HID) Demo." To run this demo, a CDC device is necessary.

In this demo, the data entered from the keyboard is echoed and displayed in HyperTerminal. The data flow in the CDC demo is shown in the following figure.

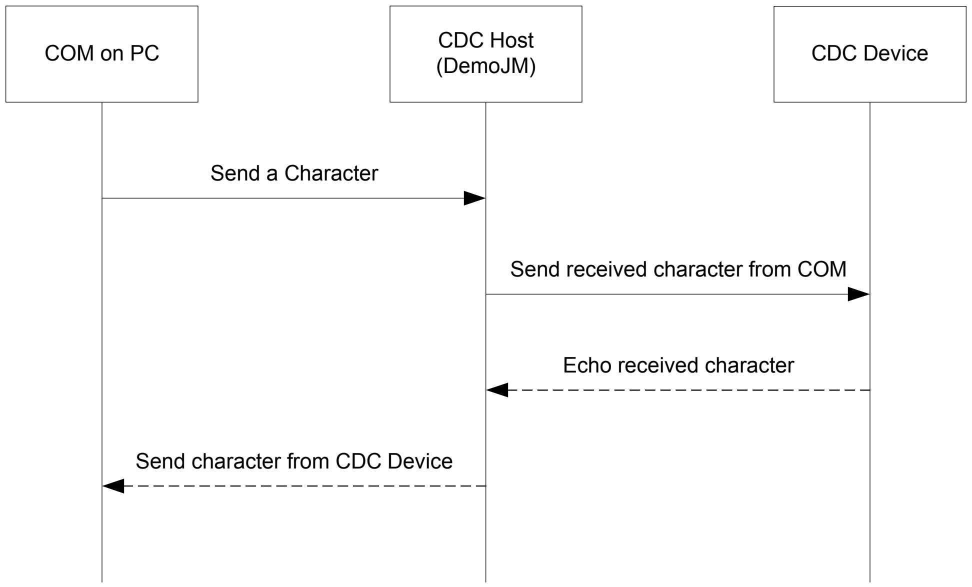


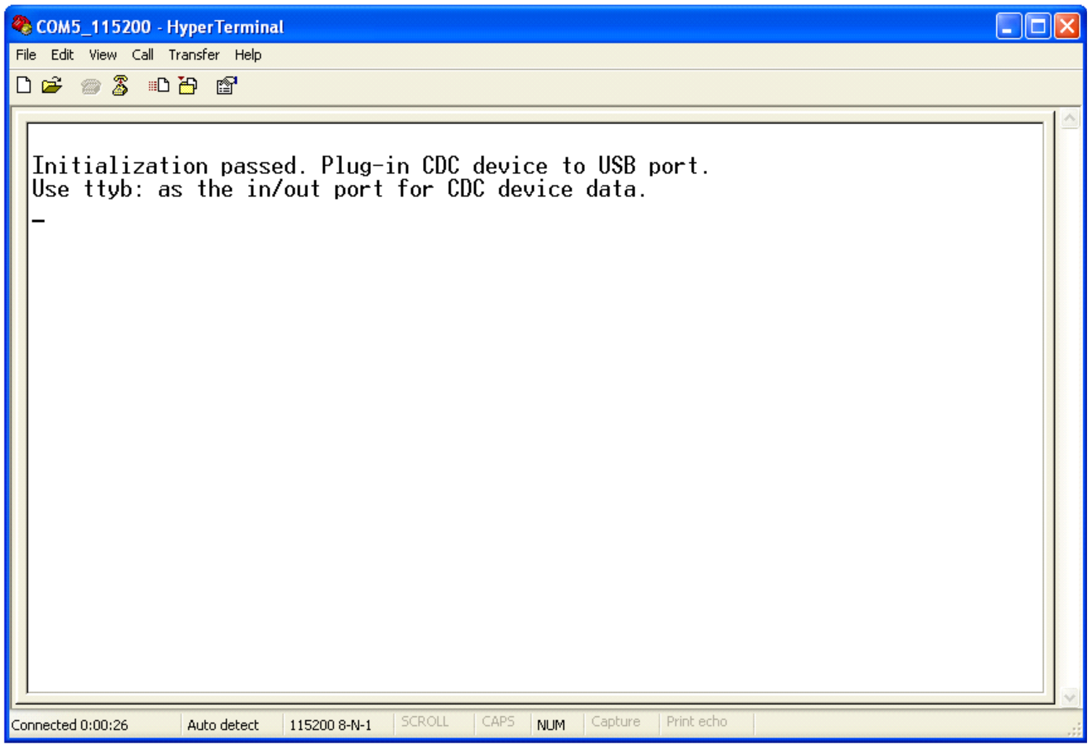
Figure C-1. Data flow

C.2 Running the demo

To run the CDC demo, perform the following steps.

1. Open the CDC demo project and load the image to the board.

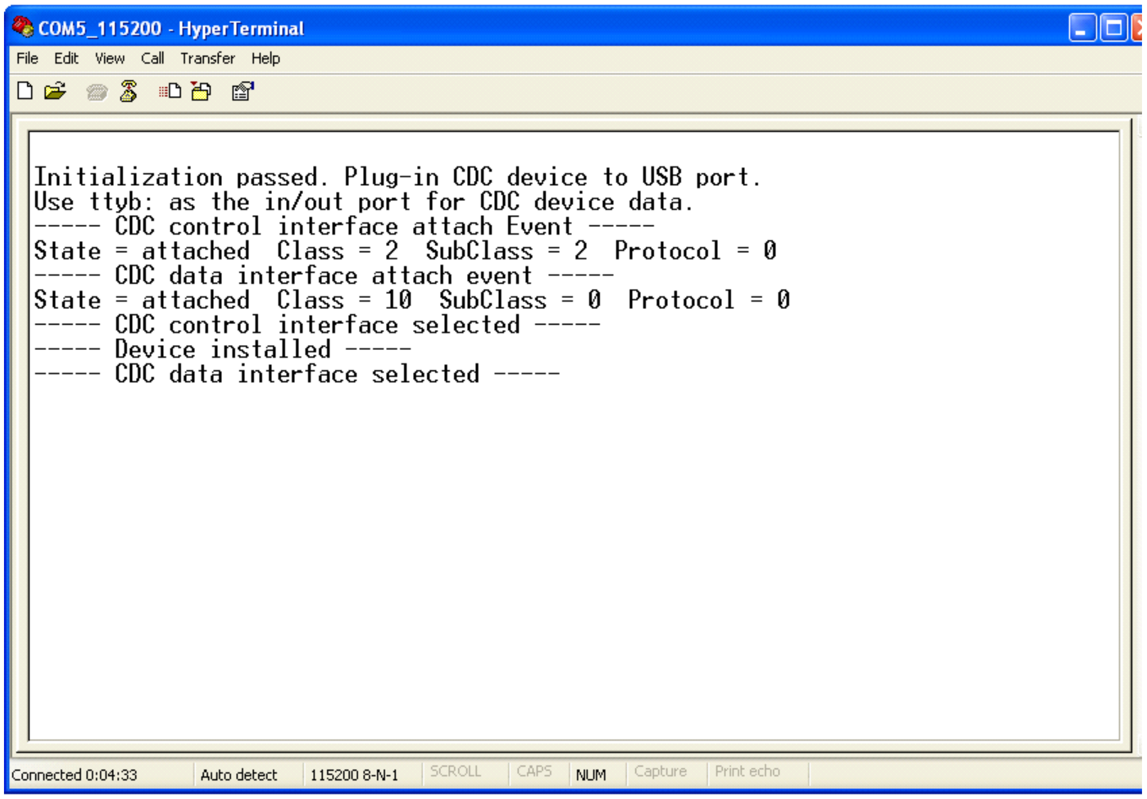
   The CDC application project is located in **\Freescale USB Stack\Source\USB Host\CDC Class Demo Apps**.
2. Connect COM1 of the board to the PC, using the steps shown in Section A.2, "Set up HyperTerminal to get log."
3. Run the demo. The HyperTerminal displays a message as shown in the following figure.



```
Initialization passed. Plug-in CDC device to USB port.
Use ttyb: as the in/out port for CDC device data.
_
```

Figure C-2. USB Host waits for CDC device plug-in

6. Plug in the CDC device to the CDC host (board). HyperTerminal shows the message seen in the following figure.

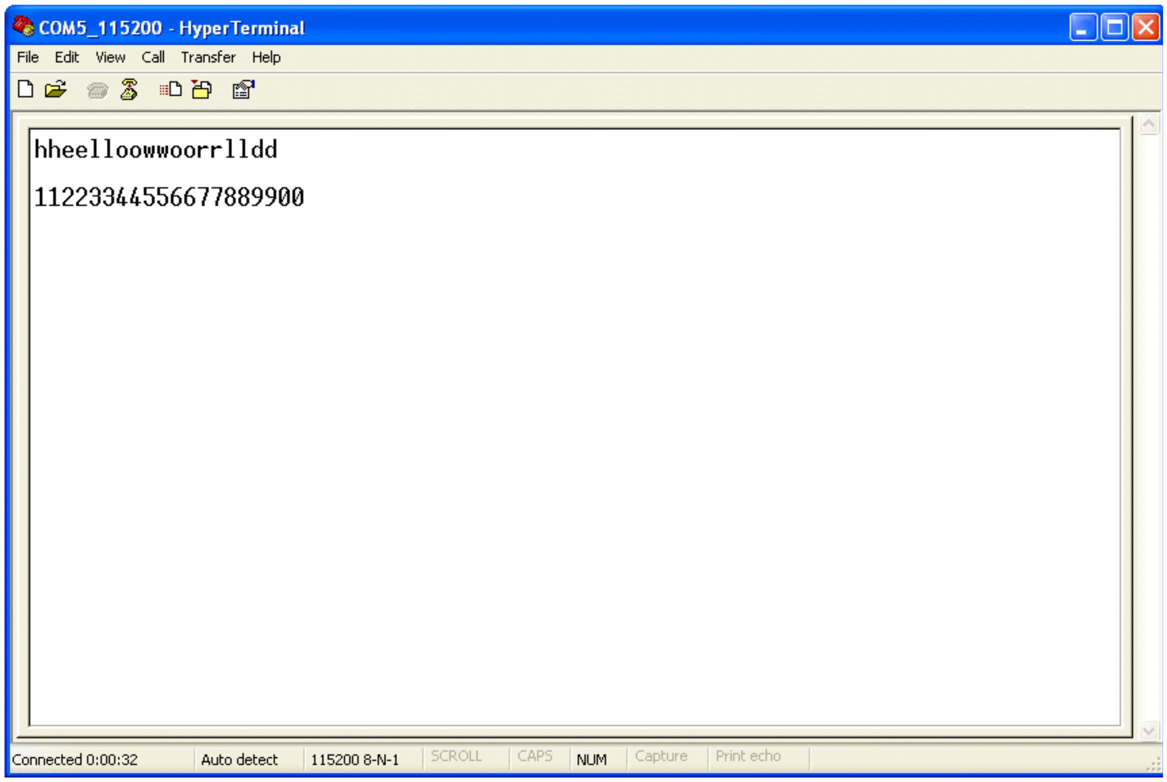

COM5_115200 - HyperTerminal
File Edit View Call Transfer Help

```
Initialization passed. Plug-in CDC device to USB port.
Use ttyb: as the in/out port for CDC device data.
----- CDC control interface attach Event -----
State = attached  Class = 2  SubClass = 2  Protocol = 0
----- CDC data interface attach event -----
State = attached  Class = 10  SubClass = 0  Protocol = 0
----- CDC control interface selected -----
----- Device installed -----
----- CDC data interface selected -----
```

Connected 0:04:33 | Auto detect | 115200 8-N-1 | SCROLL | CAPS | NUM | Capture | Print echo

Figure C-3. Device information

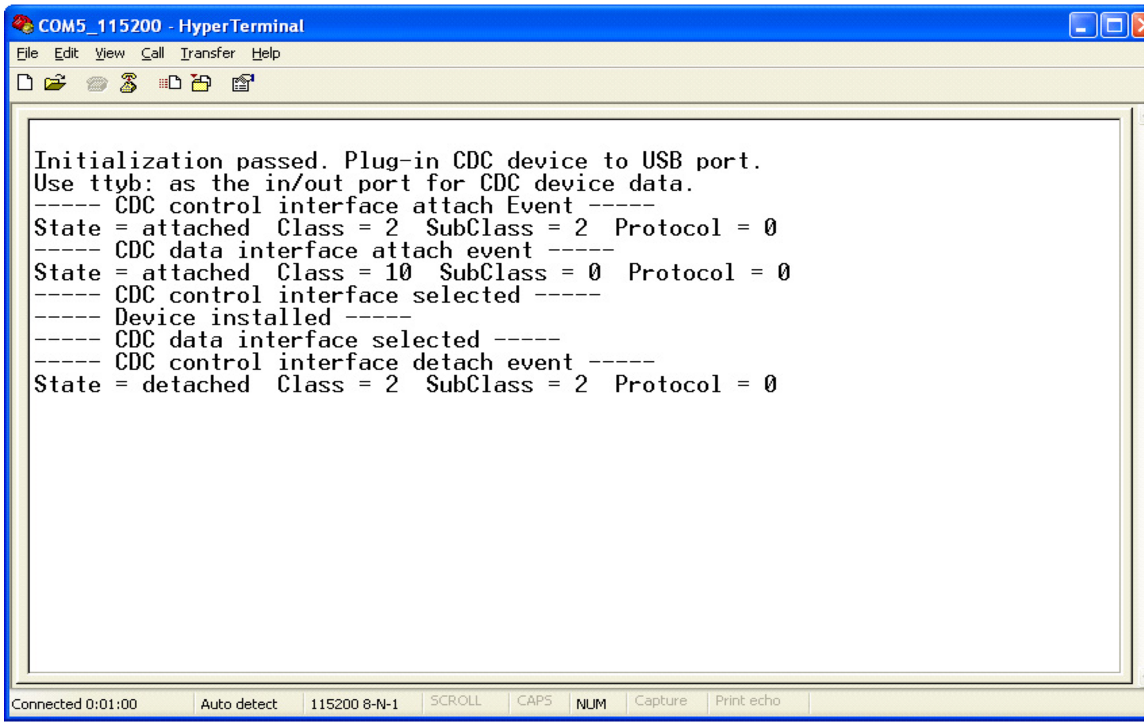
4. Disconnect COM1 from PC, connect COM2 to PC, and type something on the keyboard. The result is echoed and displayed in HyperTerminal.



```
hheelloowwoorrlldd
11223344556677889900
```

Figure C-4. Character echoed and displayed in HyperTerminal

5. Unplug the CDC device. HyperTerminal displays:



```
Initialization passed. Plug-in CDC device to USB port.
Use ttyb: as the in/out port for CDC device data.
----- CDC control interface attach Event -----
State = attached  Class = 2  SubClass = 2  Protocol = 0
----- CDC data interface attach event -----
State = attached  Class = 10  SubClass = 0  Protocol = 0
----- CDC control interface selected -----
----- Device installed -----
----- CDC data interface selected -----
----- CDC control interface detach event -----
State = detached  Class = 2  SubClass = 2  Protocol = 0
```

Figure C-5. CDC device detached

Appendix D Mass Storage Device (MSD) Demo

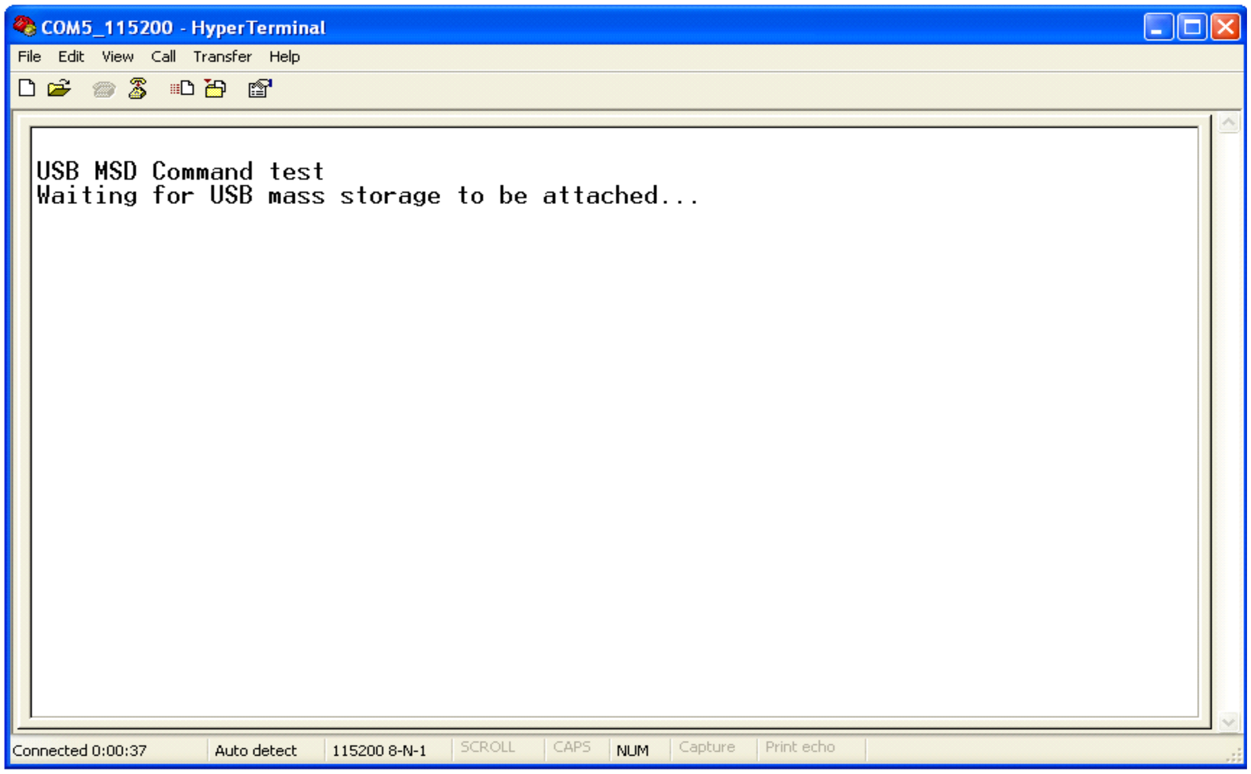
D.1 Setting up the demo

Set up the system as described in Appendix B, "Human Interface Device (HID) Demo."

D.2 Running the demo

To run this demo, perform the following steps.

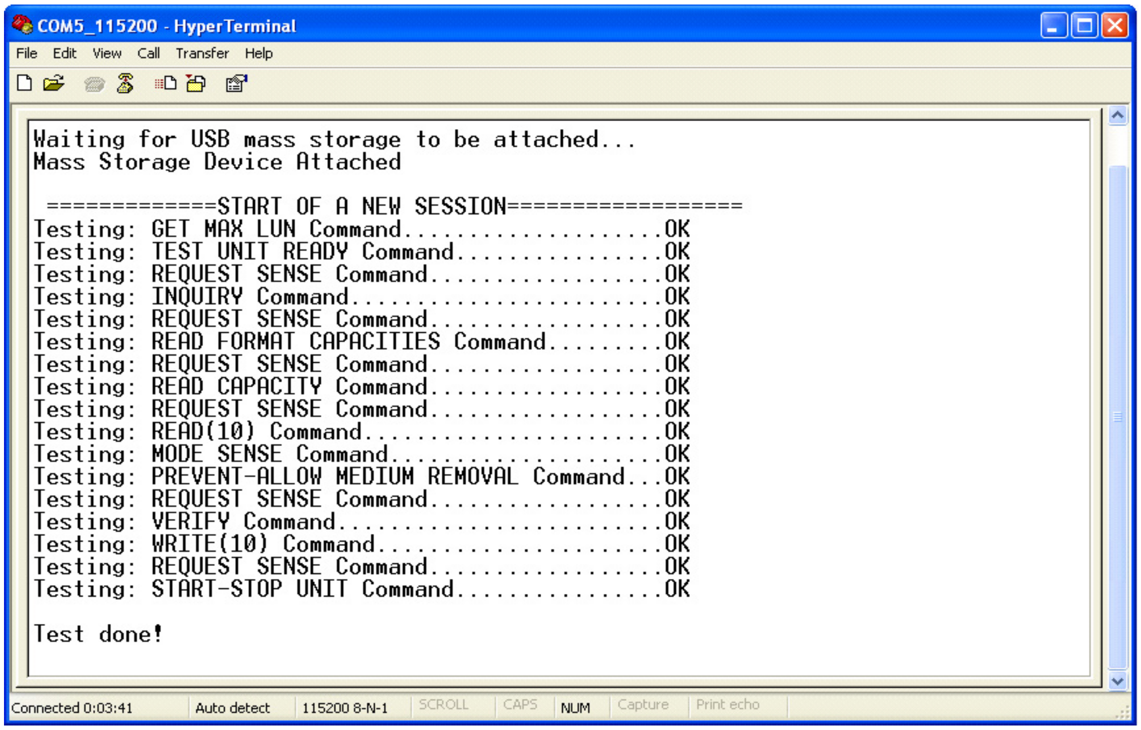
1. Open the MSD demo project and load the image to the board. The MSD application project is located in **\Freescale USB Stack\Source\USB Host\MSD Class Demo Apps**.
2. Run the demo. HyperTerminal displays a message as shown in the following figure.



```
USB MSD Command test
Waiting for USB mass storage to be attached...
```

Figure D-1. USB Host waits for USB mass storage to be attached

3. Attach USB mass storage to this board. HyperTerminal displays the test result as shown in the following figure.

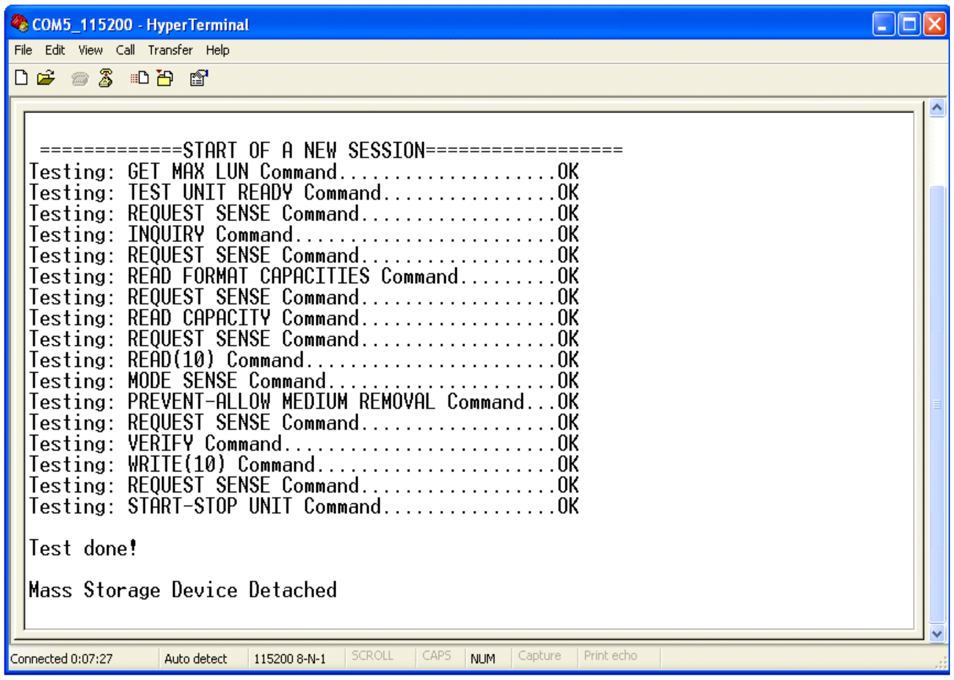


```
Waiting for USB mass storage to be attached...
Mass Storage Device Attached

 ============START OF A NEW SESSION=================
Testing: GET MAX LUN Command...................OK
Testing: TEST UNIT READY Command...............OK
Testing: REQUEST SENSE Command.................OK
Testing: INQUIRY Command.......................OK
Testing: REQUEST SENSE Command.................OK
Testing: READ FORMAT CAPACITIES Command........OK
Testing: REQUEST SENSE Command.................OK
Testing: READ CAPACITY Command.................OK
Testing: REQUEST SENSE Command.................OK
Testing: READ(10) Command......................OK
Testing: MODE SENSE Command....................OK
Testing: PREVENT-ALLOW MEDIUM REMOVAL Command...OK
Testing: REQUEST SENSE Command.................OK
Testing: VERIFY Command........................OK
Testing: WRITE(10) Command.....................OK
Testing: REQUEST SENSE Command.................OK
Testing: START-STOP UNIT Command...............OK

Test done!
```

Figure D-2. Test result

The message shows that all test cases are passed.

4. Unplug the device. HyperTerminal displays the message about device detachment as shown in the following figure.



```
 ============START OF A NEW SESSION=================
Testing: GET MAX LUN Command...................OK
Testing: TEST UNIT READY Command...............OK
Testing: REQUEST SENSE Command.................OK
Testing: INQUIRY Command.......................OK
Testing: REQUEST SENSE Command.................OK
Testing: READ FORMAT CAPACITIES Command........OK
Testing: REQUEST SENSE Command.................OK
Testing: READ CAPACITY Command.................OK
Testing: REQUEST SENSE Command.................OK
Testing: READ(10) Command......................OK
Testing: MODE SENSE Command....................OK
Testing: PREVENT-ALLOW MEDIUM REMOVAL Command...OK
Testing: REQUEST SENSE Command.................OK
Testing: VERIFY Command........................OK
Testing: WRITE(10) Command.....................OK
Testing: REQUEST SENSE Command.................OK
Testing: START-STOP UNIT Command...............OK

Test done!

Mass Storage Device Detached
```

Figure D-3. USB mass storage detached

Appendix E Audio Host Demo

This chapter is a quick guide on how to use the USB Audio Host Demo software package. The demo application is used to control and communicate with Audio Devices. The operation of the demo depends on Audio Device type:

- Speaker type (Audio Device with stream OUT supported): Sends audio data stream to the device.
- Microphone type (Audio Device with stream IN supported): Receives audio stream data from devices and play it.

In both cases, the application supports sending specific requests to the Audio Devices such as Mute Control. To take you through this guide, the demos are illustrated by using a MCF52259 Demo board.

NOTE

The Audio Host Demo supports either audio data transmit interface or audio data receive interface over isochronous pipe. In case, the Audio Devices support multi-data interfaces, the final audio data interface is supported only.

E.1 Setting up the demo

E.1.1 Hardware setup

Set up the connections as shown in Figure E-1.

1. Make the first USB connection between the PC where the software is installed and the Demo board where the silicon is mounted. This connection is required to provide power to the board and for downloading the image to the flash.
2. Make the second connection between the Demo board and the PC to display the log of the Demo board.
3. Make the third connection between the Audio Device and the Demo board.
4. Make the fourth connection between a speaker and the Audio Device. (In case Audio Device is Microphone, the speaker is connected to Audio Host instead of Audio Device).

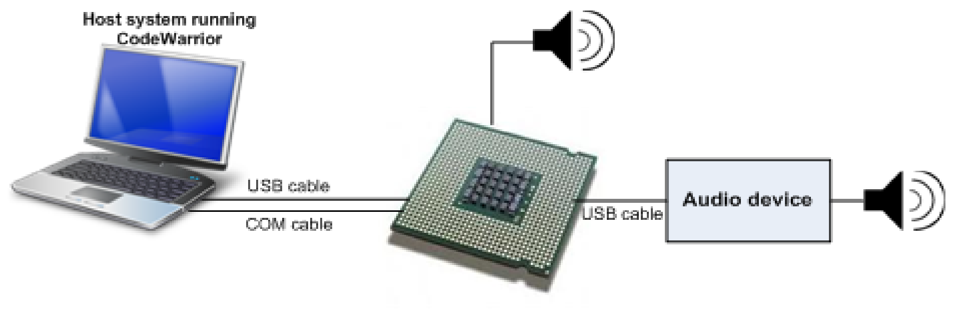


Figure E-1. Audio demo setup

E.1.2 Set up HyperTerminal to get log

To ensure that application run correctly, the HyperTerminal is used on the PC to get events from the device. These steps are used to configure HyperTerminal:

1. Open HyperTerminal applications as shown in Figure E-2

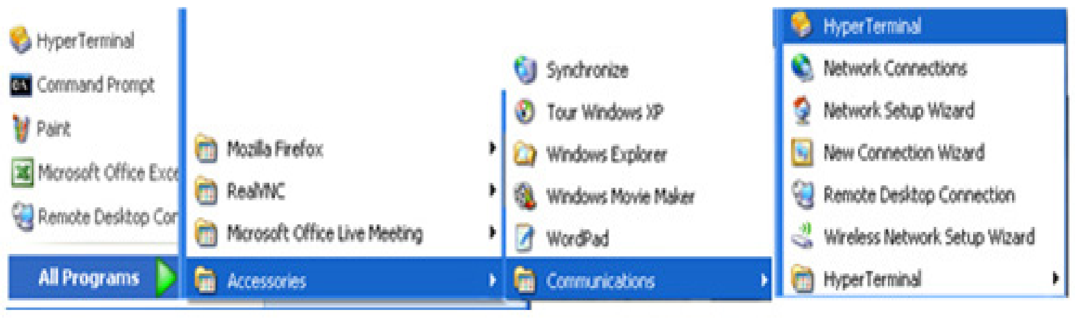
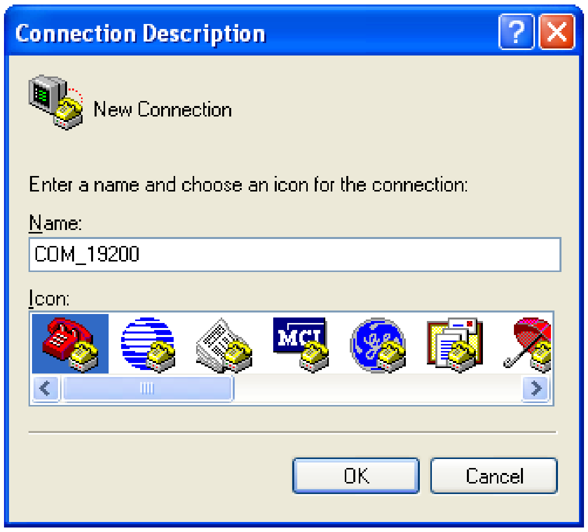


Figure E-2. Launch HyperTerminal application

2. The HyperTerminal opens as shown in Figure E-3. Enter the name of the connection and click on the OK button.

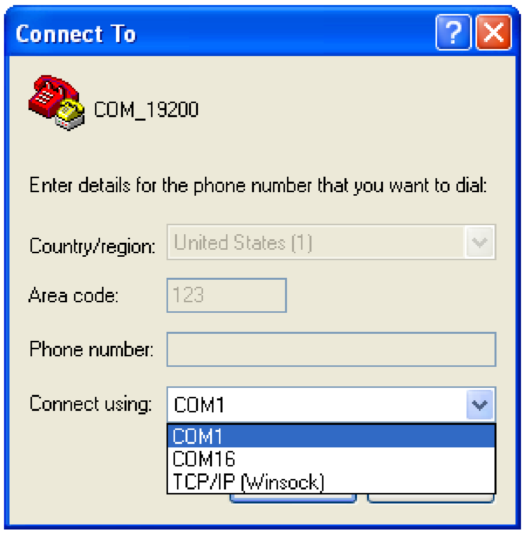


Enter a name and choose an icon for the connection:

Name: COM_19200

Figure E-3. HyperTerminal startup

3. The window shown in the following figure appears. Select the COM port.



Enter details for the phone number that you want to dial:

Country/region: United States (1)

Area code: 123

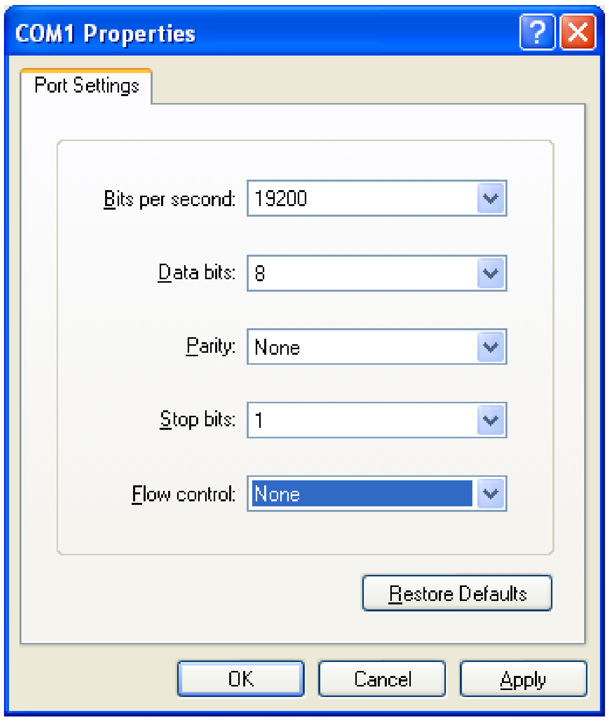
Phone number:

Connect using: COM1

COM1
COM16
TCP/IP (Winsock)

Figure E-4. Connect using COM port

4. Configure the COM port baud rate and other properties as shown in Figure E-5



COM1 Properties

Port Settings

Bits per second: 19200

Data bits: 8

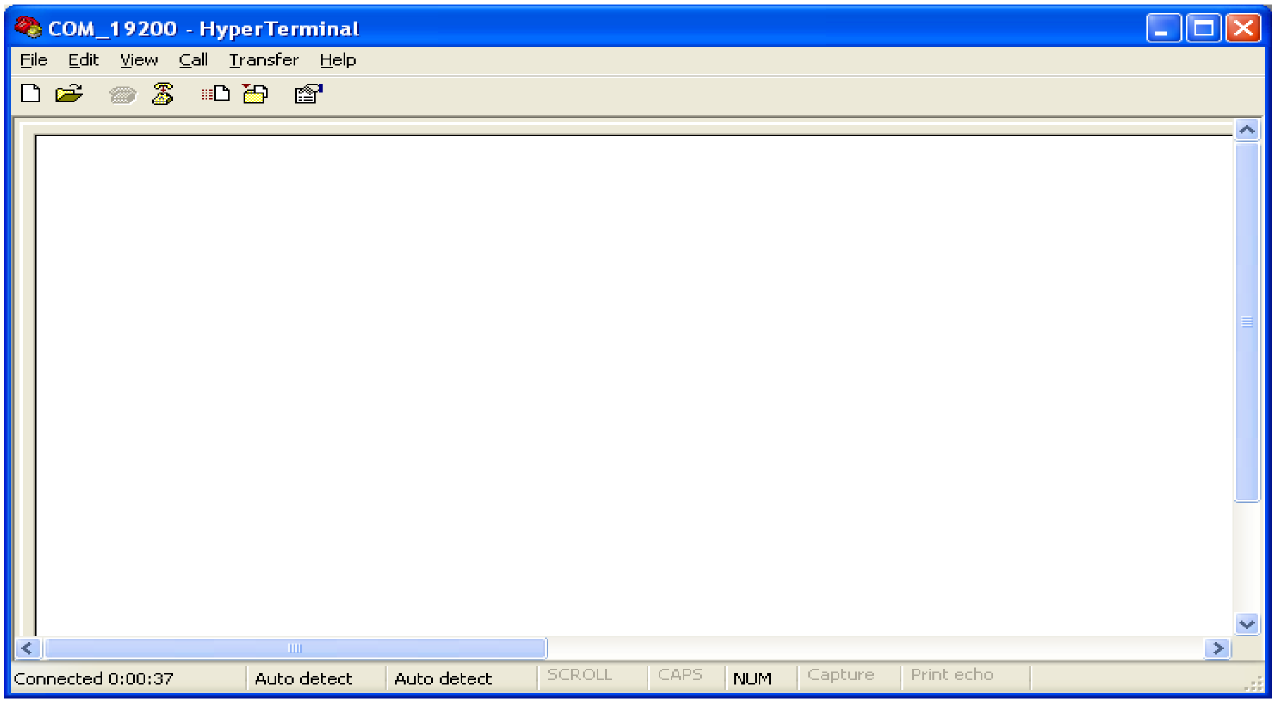
Parity: None

Stop bits: 1

Flow control: None

Figure E-5. COM properties

5. The HyperTerminal is now configured as shown in Figure E-6



COM_19200 - HyperTerminal

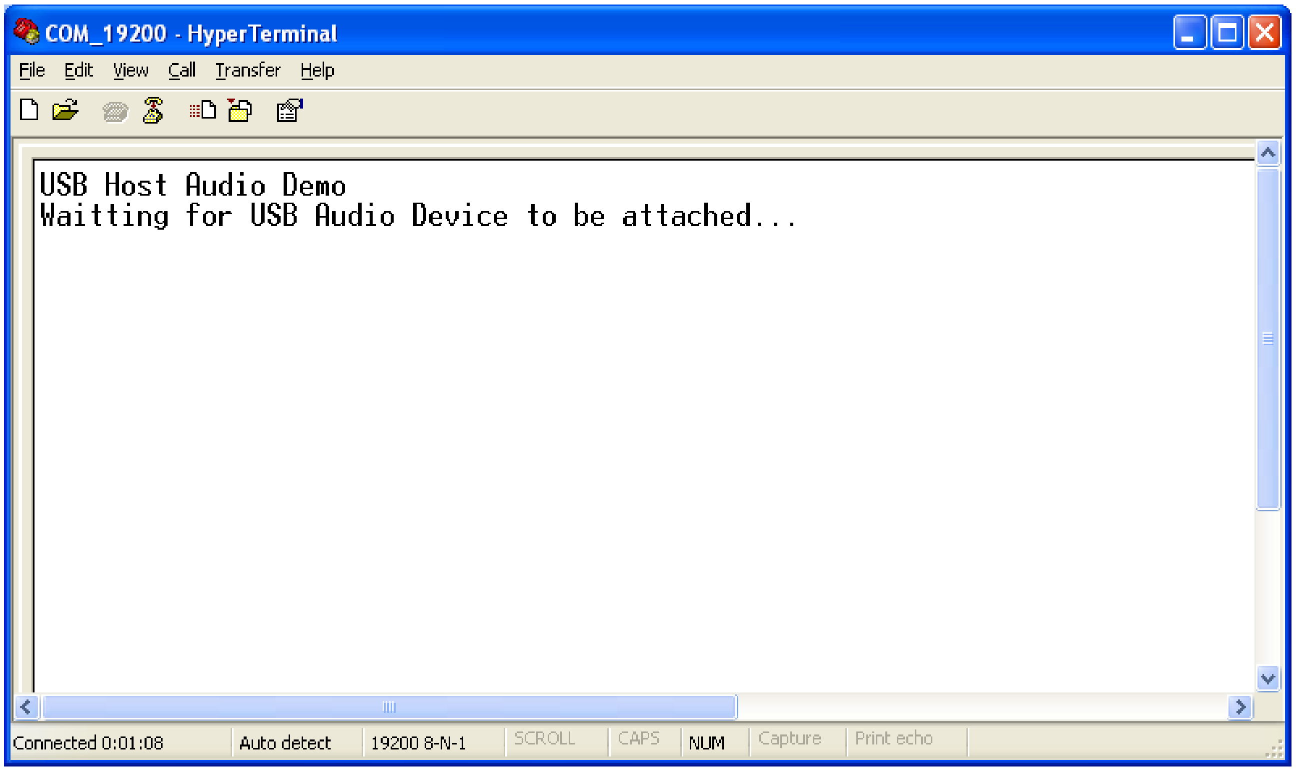
Connected 0:00:37 | Auto detect | Auto detect | SCROLL | CAPS | NUM | Capture | Print echo

Figure E-6. HyperTerminal

E.1.3 Running the demo

Perform the following steps to run the Audio Host Demo:

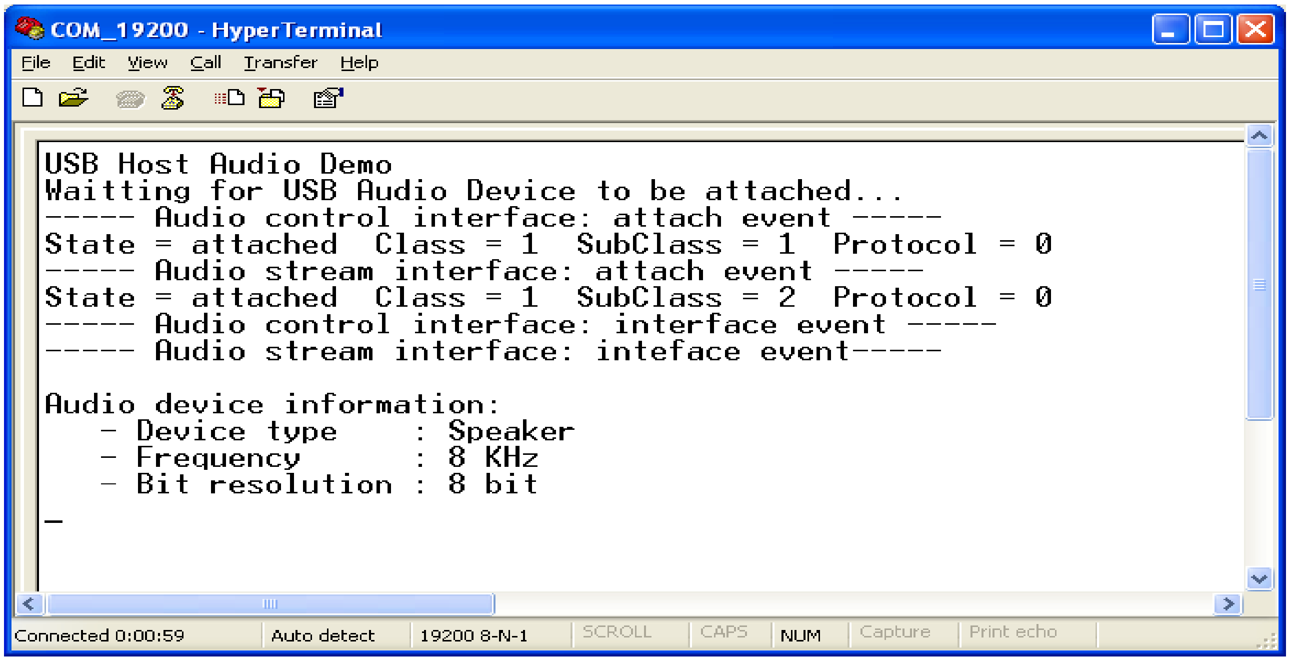
1. Open and load the image of Audio Demo application to the board.
2. After the image has been loaded successfully, HyperTerminal appears as shown in Figure 2 7.



```
USB Host Audio Demo
Waitting for USB Audio Device to be attached...
```

Figure E-7. USB Host waiting for audio device attachment event

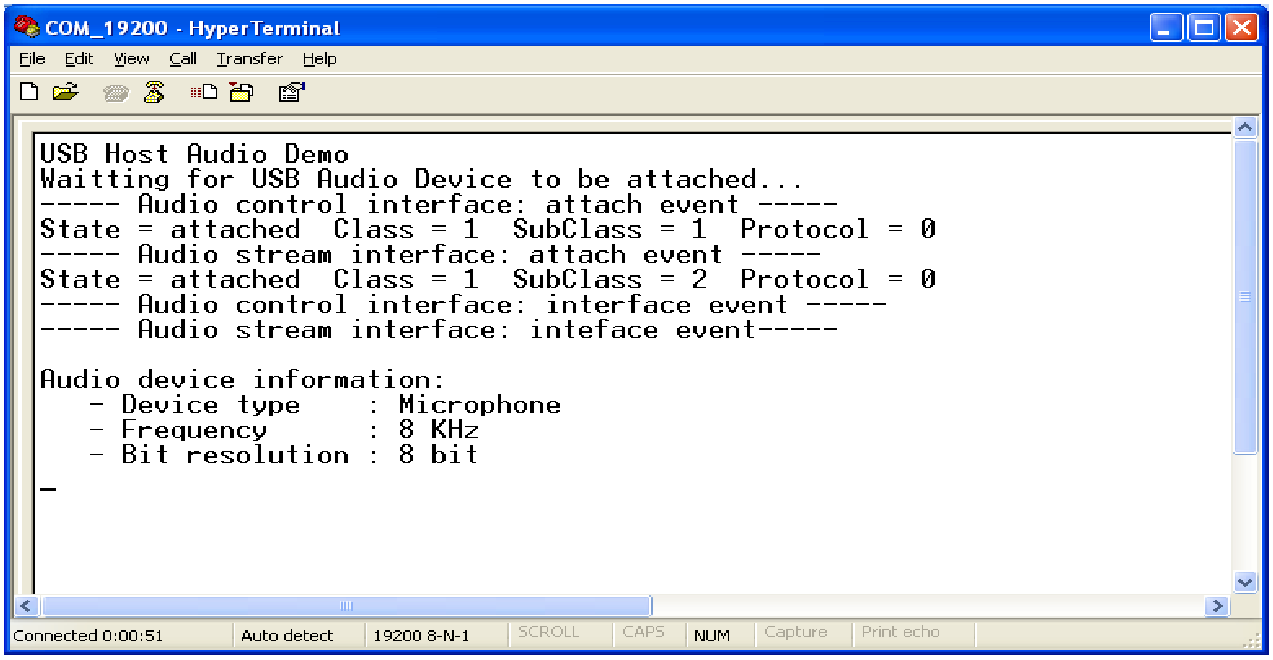
3. Plug the Audio Device into the board. The Audio Device will be attached. Device information is shown as in Figure E-8 if Audio Device is Speaker type, in Figure E-9 if Audio Device is Microphone type.



```
USB Host Audio Demo
Waitting for USB Audio Device to be attached...
----- Audio control interface: attach event -----
State = attached  Class = 1  SubClass = 1  Protocol = 0
----- Audio stream interface: attach event -----
State = attached  Class = 1  SubClass = 2  Protocol = 0
----- Audio control interface: interface event -----
----- Audio stream interface: inteface event-----

Audio device information:
   - Device type     : Speaker
   - Frequency       : 8 KHz
   - Bit resolution  : 8 bit
_
```

```
Connected 0:00:59    Auto detect    19200 8-N-1    SCROLL    CAPS    NUM    Capture    Print echo
```

Figure E-8. Attached device is Speaker type

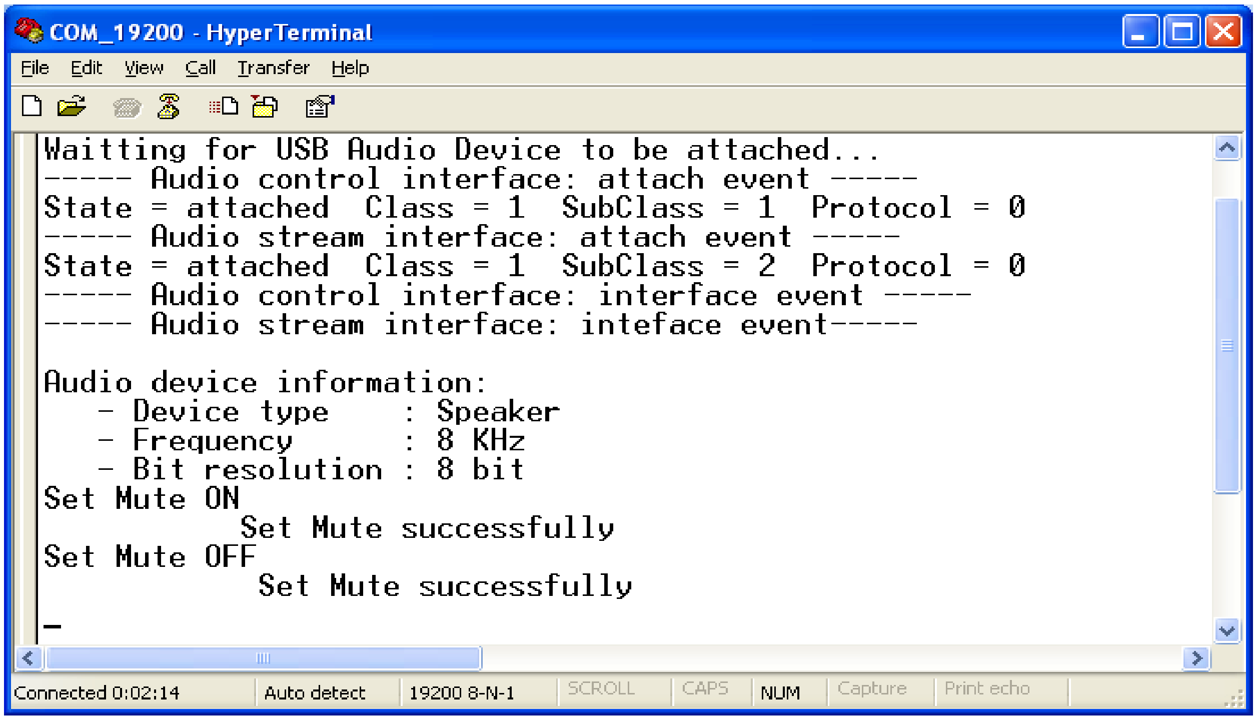


```
USB Host Audio Demo
Waitting for USB Audio Device to be attached...
----- Audio control interface: attach event -----
State = attached  Class = 1  SubClass = 1  Protocol = 0
----- Audio stream interface: attach event -----
State = attached  Class = 1  SubClass = 2  Protocol = 0
----- Audio control interface: interface event -----
----- Audio stream interface: inteface event-----

Audio device information:
   - Device type     : Microphone
   - Frequency       : 8 KHz
   - Bit resolution  : 8 bit
_
```

```
Connected 0:00:51    Auto detect    19200 8-N-1    SCROLL    CAPS    NUM    Capture    Print echo
```

Figure E-9. Attached device is Microphone type

4. Press Switch 2 to set Mute ON/OFF. The HyperTerminal screen appears as shown in Figure E-10.

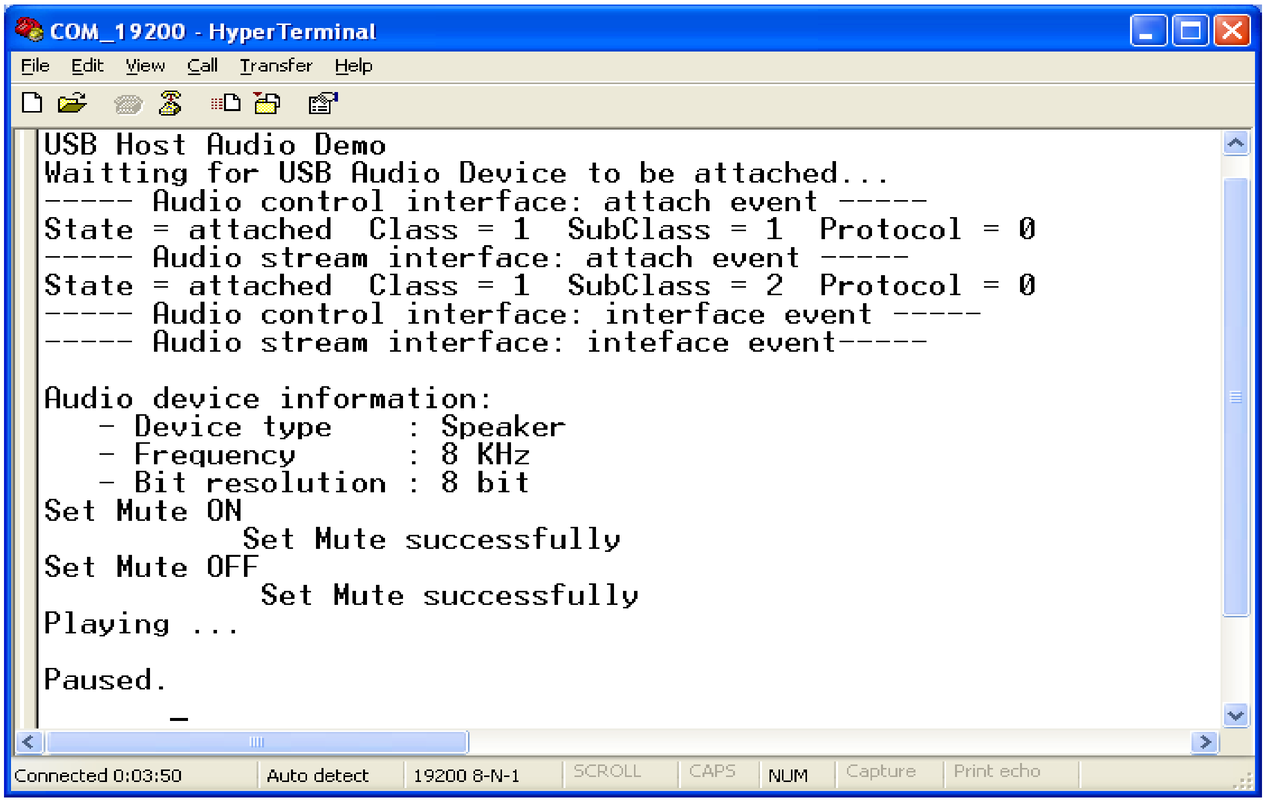


```
Waitting for USB Audio Device to be attached...
----- Audio control interface: attach event -----
State = attached  Class = 1  SubClass = 1  Protocol = 0
----- Audio stream interface: attach event -----
State = attached  Class = 1  SubClass = 2  Protocol = 0
----- Audio control interface: interface event -----
----- Audio stream interface: inteface event-----

Audio device information:
   - Device type    : Speaker
   - Frequency      : 8 KHz
   - Bit resolution : 8 bit
Set Mute ON
           Set Mute successfully
Set Mute OFF
            Set Mute successfully
_
```

Figure E-10. Set Mute ON/OFF

5. Press Switch 1 to Start/Stop transferring audio data stream between the Audio Host and the Audio Device.
   – If attached device is Speaker type, you can hear the sound from the speaker, which is connected to the Audio Device.
   – If attached device is Microphone type, you can hear the sound from the speaker, which is connected to the Audio Host.

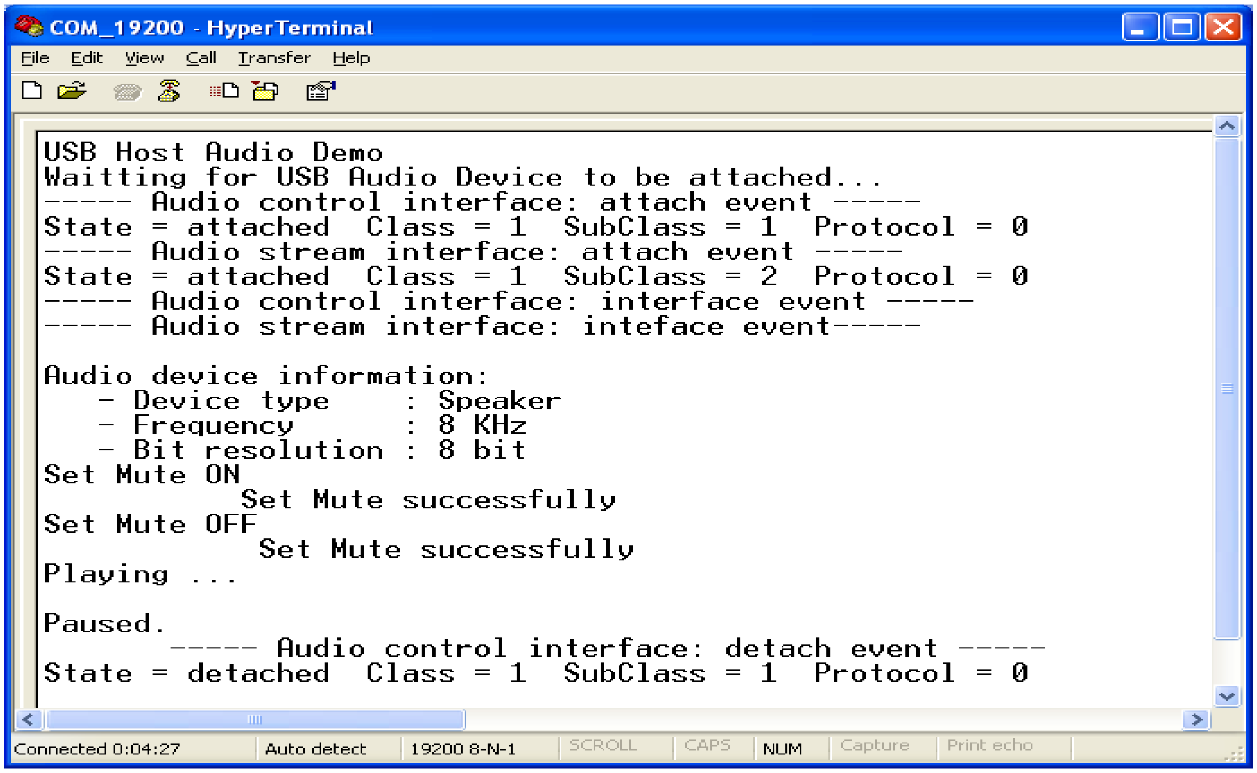
The HyperTerminal screen appears as shown in Figure E-11



```
USB Host Audio Demo
Waitting for USB Audio Device to be attached...
----- Audio control interface: attach event -----
State = attached  Class = 1  SubClass = 1  Protocol = 0
----- Audio stream interface: attach event -----
State = attached  Class = 1  SubClass = 2  Protocol = 0
----- Audio control interface: interface event -----
----- Audio stream interface: inteface event-----

Audio device information:
   - Device type    : Speaker
   - Frequency      : 8 KHz
   - Bit resolution : 8 bit
Set Mute ON
           Set Mute successfully
Set Mute OFF
            Set Mute successfully
Playing ...

Paused.
       _
```

Figure E-11. Start/Stop transferring audio data

6. Unplug the Audio Device. The HyperTerminal shows a message as shown in Figure E-12



```
USB Host Audio Demo
Waitting for USB Audio Device to be attached...
----- Audio control interface: attach event -----
State = attached  Class = 1  SubClass = 1  Protocol = 0
----- Audio stream interface: attach event -----
State = attached  Class = 1  SubClass = 2  Protocol = 0
----- Audio control interface: interface event -----
----- Audio stream interface: inteface event-----

Audio device information:
   - Device type     : Speaker
   - Frequency       : 8 KHz
   - Bit resolution : 8 bit
Set Mute ON
           Set Mute successfully
Set Mute OFF
            Set Mute successfully
Playing ...

Paused.
       ----- Audio control interface: detach event -----
State = detached  Class = 1  SubClass = 1  Protocol = 0
```

Figure E-12. Audio Device detached

Appendix F USB FAT File System Demo

The demo application demonstrates how to use application interface functions of the FATFS module to operate with file and directory of mass storage devices.

F.1 Setting up the demo

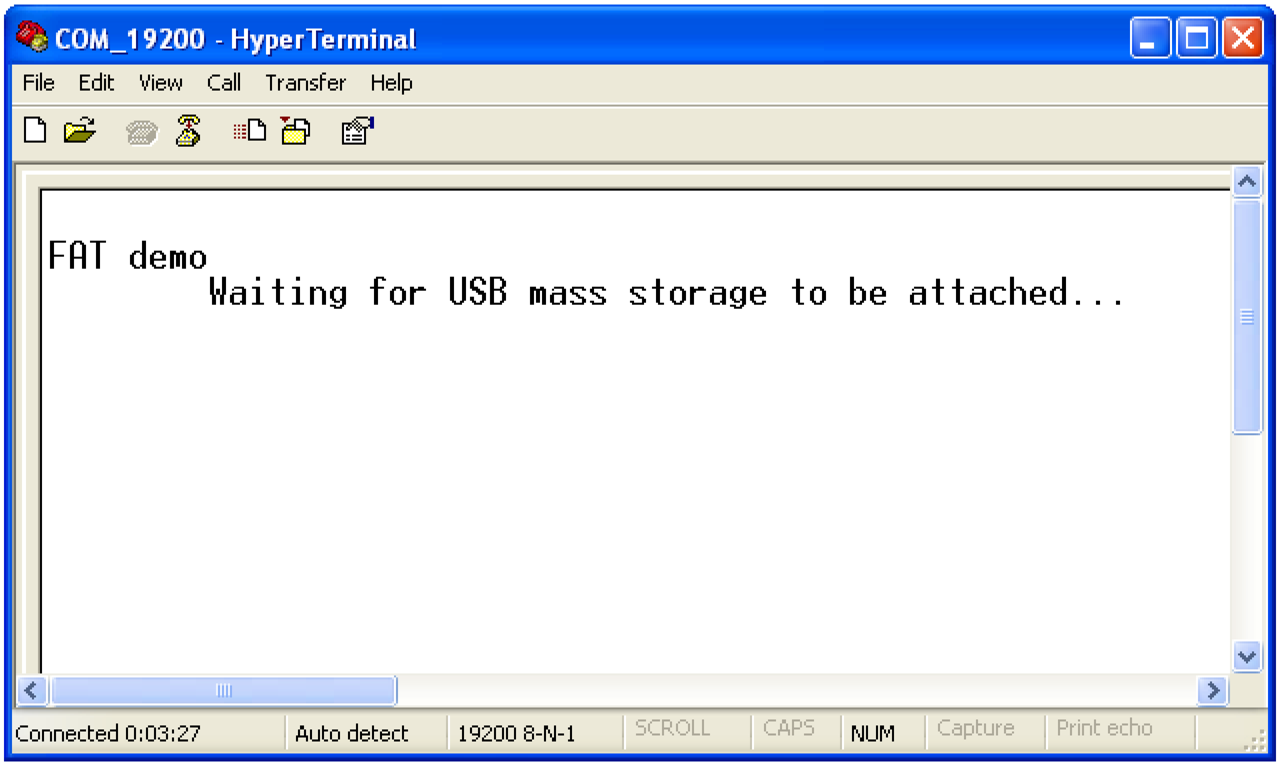
Set the system as described in the Section A.1.1.2, "Hardware setup."

F.2 Running the demo

F.2.1 Mouse demo

Perform the following steps to run the mouse demo:

1. Open and load the image of USB FATFS Demo application to the board.
2. After the image has been loaded successfully, the HyperTerminal appears as shown in Figure F-1.



```
FAT demo
        Waiting for USB mass storage to be attached...
```

Figure F-1. The USB Host is waiting the mass storage device attachment event

3. Plug an USB Mass Storage Device into the board. The Mass Storage Device will be attached and all functionalities of FATFS are implemented, sequentially and the results are shown in the HyperTerminal. The detail of display content is shown as following:

```
FAT demo
Waiting for USB mass storage to be attached...
Mass Storage Device Attached
******************************************************************************
*                        FATfs DEMO                              *
*         Configuration:  LNF Enabled, Code page  =1258                    *
******************************************************************************
******************************************************************************
*                        DRIVER OPERATION                              *
******************************************************************************
1. Demo funciton: f_mount

  Initializing logical drive 0...
  Initialization complete
--------------------------------------------------------------------------


2. Demo funcitons:f_getfree, f_opendir, f_readdir

getting drive 0 attributes...............
Logical drive 0 attributes:
 FAT type = FAT32
 Bytes/Cluster = 512
 Number of FATs = 2
 Root DIR entries = 0
Sectors/FAT = 618
 Number of clusters = 78931
 FAT start (lba) = 158
 DIR start (lba,clustor) = 623
 Data start (lba) = 1394


...
39465 KB total disk space.
25666 KB available.
--------------------------------------------------------------------------
FAT type = FAT32
 Bytes/Cluster = 512
 Number of FATs = 2
Root DIR entries = 0
```

```
 Sectors/FAT = 618
 Number of clusters = 78931
 FAT start (lba) = 158
 DIR start (lba,clustor) = 623
 Data start (lba) = 1394

...
39465 KB total disk space.
25666 KB available.
-----------------------------------------------------------------------------
*****************************************************************************
*                          DRECTORY OPERATION                               *
*****************************************************************************
1. Demo funcitons:f_opendir, f_readdir

 Directory listing...
   D---- 2010/12/23 15:41         0  New Folder
   DR--- 2010/12/25 23:30         0  Directory_1
   ----A 2010/12/23 15:42        33  dsgsgsg.dat
   D---- 2010/01/01 00:00         0  Directory_2
   ----A 2010/01/01 00:00        32  file_test.txt
   ----A 2010/12/28 16:26   1307648  FSL_USB_MSD_FATFS_Development_Design_v1.1.doc
   ----A 2010/12/09 08:43    826338  ff8a.zip
   D-HS- 2010/12/28 18:12         0  Recycled
   D-HS- 2010/12/28 18:12         0  System Volume Information
   ----A 2010/12/28 10:19    302592  FSL_USB_MSD_FATFS_Demo_SDD_V1.1.doc
   D---- 2010/12/28 18:19         0  Freescale USB Stack v2.6
   ----A 2010/12/30 17:52     65024  FSL_USB_MSD_FAT_Development_System Test 
Case.v0.1.xls
   ----A 2010/12/29 19:15    477734  2010_12_29_MSD_FATFS_Source_Code.zip
   ----A 2010/12/29 16:23   3880022  fat-2006-12-03.zip



   8    File(s),   6859423 bytes total
   6    Dir(s)
-----------------------------------------------------------------------------
2. Demo funcitons:f_mkdir

2.0. Create <Directory_1>
2.1. Create <Directory_2>
2.2. Create <Sub1> as a sub directory of <Directory_1>
2.3. Directory list
Directory listing...
```

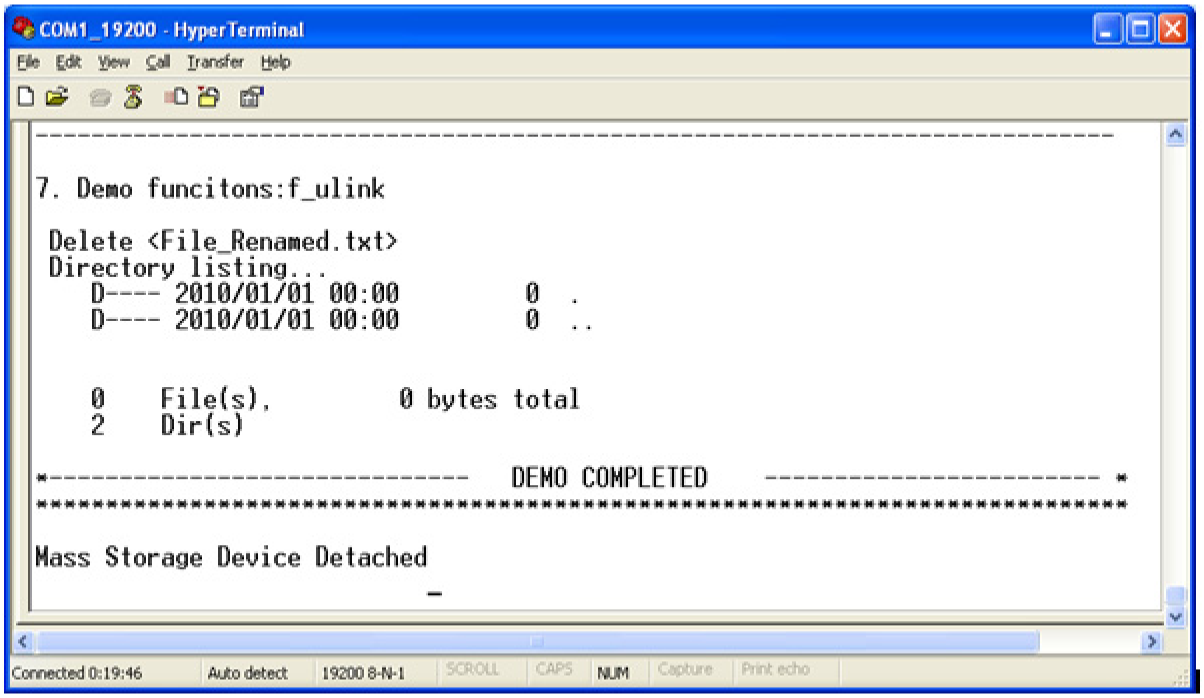
```
 D---- 2010/12/23 15:41        0  New Folder
   DR--- 2010/12/25 23:30        0  Directory_1
   ----A 2010/12/23 15:42       33  dsgsgsg.dat
   D---- 2010/01/01 00:00        0  Directory_2
   ----A 2010/01/01 00:00       32  file_test.txt
   ----A 2010/12/28 16:26   1307648  FSL_USB_MSD_FATFS_Development_Design_v1.1.doc
   ----A 2010/12/09 08:43    826338  ff8a.zip
   D-HS- 2010/12/28 18:12        0  Recycled
   D-HS- 2010/12/28 18:12        0  System Volume Information
   ----A 2010/12/28 10:19    302592  FSL_USB_MSD_FATFS_Demo_SDD_V1.1.doc
   D---- 2010/12/28 18:19        0  Freescale USB Stack v2.6
   ----A 2010/12/30 17:52     65024  FSL_USB_MSD_FAT_Development_System Test
Case.v0.1.xls
   ----A 2010/12/29 19:15    477734  2010_12_29_MSD_FATFS_Source_Code.zip
   ----A 2010/12/29 16:23   3880022  fat-2006-12-03.zip



   8    File(s),    6859423 bytes total
   6    Dir(s)
------------------------------------------------------------------------
3. Demo funcitons:f_getcwd, f_chdir

3.0. Get the current directory
   CWD: 0:/
3.1. Change current directory to <Directory_1>
3.2. Directory listing
 Directory listing...
   D---- 2010/01/01 00:00        0  .
   D---- 2010/01/01 00:00        0  ..
   D---- 2010/01/01 00:00        0  sub1



   0    File(s),         0 bytes total
   3    Dir(s)
3.3. Get the current directory
   CWD: 0:/Directory_1
------------------------------------------------------------------------


4. Demo funcitons:f_stat(File status), f_chmod, f_utime

4.1. Get directory information of <Directory_1>
   DR--- 2010/12/25 23:30        0  DIRECT~1
4.2   Change the timestamp of Directory_1 to 12.25.2010: 23h 30' 20
```

```
4.3. Set Read Only Attribute to Directory_1
4.4. Get directory information (Directory_1)
   DR--- 2010/12/25 23:30        0  DIRECT~1
--------------------------------------------------------------------------
5. Demo funcitons:f_rename

Rename <sub1> to <sub1_renamed> and move it to <Directory_2>
 Directory listing...
   D---- 2010/01/01 00:00        0  .
   D---- 2010/01/01 00:00        0  ..
   D---A 2010/01/01 00:00        0  sub1_renamed


   0    File(s),        0 bytes total
   3    Dir(s)
--------------------------------------------------------------------------

6. Demo funcitons:f_unlink

 Delete Directory_1/sub1_renamed
 Directory listing...
   D---- 2010/01/01 00:00        0  .
   D---- 2010/01/01 00:00        0  ..


   0    File(s),        0 bytes total
   2    Dir(s)

******************************************************************************
*                        FILE OPERATION                         *
******************************************************************************
1. Demo funcitons:f_open,f_write, f_printf, f_putc, f_puts, fclose

1.0. Create new file <New_File_1> (f_open)
   File size =    0
1.1. Write data to <New_File_1>(f_write)
1.2. Flush cached data
   File size =   52
1.3. Write data to <New_File_1> (f_printf)
1.4. Flush cached data
   File size =  103
1.5. Write data to <New_File_1> (f_puts)
```

```
1.6. Flush cached data
   File size =  152
1.7. Write data to <New_File_1> uses f_putc function
1.8. Flush cached data
   File size =  199
1.9. Close file <New_File_1>
------------------------------------------------------------------------
2. Demo funcitons:f_open,f_read, f_seek, f_gets, f_close


2.0. Open <New_File_1> to read (f_open)
2.1. Get a string from file (f_gets)
   Line 1: Write data to  file uses f_write function
2.2 Get the rest of file content (f_read)
ine 2: Write data to file uses f_printf function
Line 3: Write data to file uses f_puts function
Line 4: Write data to file uses f_putc function Š
2.2. Close file (f_close)
------------------------------------------------------------------------


2. Demo funcitons:f_stat, f_utime, f_chmod

3.1. Get  information of <New_File_1> file (f_stat)
   ----A 2010/01/01 00:00       199  NEW_FI~1.DAT
3.2  Change the timestamp of Directory_1 to 12.25.2010: 23h 30' 20 (f_utime)
3.3. Set Read Only Attribute to <New_File_1> (f_chmod)
3.4. Get directory information of <New_File_1> (f_stat)
   -R--A 2010/12/25 23:30       199  NEW_FI~1.DAT
3.5. Clear Read Only Attribute of <New_File_1> (f_chmod)
3.6. Get directory information of <New_File_1>
   ----A 2010/12/25 23:30       199  NEW_FI~1.DAT
------------------------------------------------------------------------
4. Demo funcitons:f_ulink

 Rename <New_File_1.dat> to  <File_Renamed.txt>
 Directory listing...
   D---- 2010/01/01 00:00         0  .
   D---- 2010/01/01 00:00         0  ..
   ----A 2010/12/25 23:30       199  File_Renamed.txt



1    File(s),      199 bytes total
```

```
 2    Dir(s)
-----------------------------------------------------------------------------
5. Demo funcitons:f_truncate

 Truncate file <File_Renamed.txt>
5.0. Open <File_Renamed.txt> to write
5.1. Seek file pointer
    Current file pointer:    0
    File pointer affter seeking:  102
5.2. Truncate file
    File size =  102
5.3. Close file
-----------------------------------------------------------------------------


6. Demo funcitons:f_forward

6.0. Open <File_Renamed.txt> to read
6.1. Forward file to ternimal
Line 1: Write data to  file uses f_write function
Line 2: Write data to file uses f_printf function
6.2. Close file
-----------------------------------------------------------------------------
7. Demo funcitons:f_ulink

 Delete <File_Renamed.txt>
 Directory listing...
    D---- 2010/01/01 00:00        0  .
    D---- 2010/01/01 00:00        0  ..



    0    File(s),        0 bytes total
    2    Dir(s)

*------------------------------   DEMO COMPLETED    ------------------------ *
*******************************************************************************
```

4. Unplug mouse from board. The HyperTerminal shows a message as shown in Figure F-2.



```
-----------------------------------------------------------------------------

7. Demo funcitons:f_ulink

 Delete <File_Renamed.txt>
 Directory listing...
    D---- 2010/01/01 00:00         0  .
    D---- 2010/01/01 00:00         0  ..


    0    File(s),         0 bytes total
    2    Dir(s)

*------------------------------    DEMO COMPLETED    ------------------------ *
******************************************************************************

Mass Storage Device Detached
                                             _
```

Figure F-2. Mass storage device detached

F.3 FATFS Test Application

The test application is used to verify whether or not application interface functions of the FAT module work properly.

F.3.1 Setting up the demo

Set the system as described in the Section A.1.1.2, "Hardware setup."

F.3.2 Running the demo

Steps to run test application are similar to demo application described in Section B.2, "Running the demo."

NOTE

Make sure that your USB mass storage device under test is divided into two partitions which do not contain any data.

There are some test cases that need special setting in FATFS module configuration (ffconf.h), so test case set is divided into three exclusive running groups:

1. Test group 1
2. Test group 2
3. Test group 3

F.3.2.1 Test Group 1

The test group 1 contains the following subgroups:

Table 5-4. Test group 1

Subgroup	Description	FATFS module configuration
TestDir1	This test group is to test f_mkdir, f_unlink functions with 0 of recursive level	#define _FS_TINY 1 #define _FS_READONLY 0 #define _FS_MINIMIZE 0 #define _USE_STRFUNC 1 #define _USE_FORWARD 1 #define _USE_LFN 3 #define _MAX_LFN 255 #define _FS_RPATH 2 #define _MULTI_PARTITION 0 #define _VOLUMES 1
TestDir2	This test group is to test f_mkdir in cases of invalid directory names	
TestDir3	This test group is to test f_unlink in cases of invalid directory names	
TestDir4	This test group is to test f_mkdir, f_unlink functions with 1 of recursive level	
TestDir5	This test group is to test f_mkdir, f_unlink functions with 2 of recursive level	
TestDir6	This test group is to test f_chdir, f_getcwd, f_unlink functions	
TestDir7	This test group is to test f_mkdir and f_unlink many of sub-directories	
TestDir8	This test group is to test f_opendir, f_readdir functions	
TestDir9	This test group is to test f_chdir function with ".." directory	
TestDir10	This test group is to test f_readdir in case of there are many files in read directory	
TestDir11	This test group is to test f_stat, f_utime, f_chmod functions	
TestFile1	This test group is to test f_open, f_close, and f_unlink functions	
TestFile2	This test group is to test f_write and f_read functions	
TestFile3	This test group is to test f_lseek function	
TestFile4	This test group is to test f_stat, f_utime, f_chmod functions	
TestFile5	This test group is to test f_forward function	
TestFile6	This test group is to test f_truncate function	
TestFile7	This test group is to test f_sync function	
TestFile8	This test group is to test string functions	
TestDirFileMixup1	This test group is to test mix file and directory	

To enable the test group, define the macro **RUN_TEST_101_111_201_209_301** in file **testcase.h**. Subgroups **TestDir7, TestDir8,** and **TestDir10** contain test cases that make or create a lot of directories and files. It takes long time, if created, the number of directories and files is large. How many directories and files will be created is specified by macro **NUM_REPEAT** in **testcase.h** file.

Expected results of these test cases are shown in the HyperTerminal as follows.

```
FAT test
Waiting for USB mass storage to be attached...
Mass Storage Device Attached
Test Cases:
  101: Test Directory Functions - 1: f_mkdir, f_unlink functions with 0 of recursive level.
  102: Test Directory Functions - 2: f_mkdir in cases of invalid directory names.
  103: Test Directory Functions - 3: f_unlink in cases of invalid directory names.
  104: Test Directory Functions - 4: f_mkdir, f_unlink functions with 1 of recursive level.
  105: Test Directory Functions - 5: f_mkdir, f_unlink functions with 2 of recursive level.
  106: Test Directory Functions - 6: f_chdir, f_getcwd, f_unlink functions.
  107: Test Directory Functions - 7: Make maximum number of sub-directories.
  108: Test Directory Functions - 8: f_opendir, f_readdir functions.
  109: Test Directory Functions - 9: f_chdir function with .. directory.
  110: Test Directory Functions - 10: f_readdir in case of there are many files in read directory.
  111: Test Directory Functions - 11: f_stat, f_utime, f_chmod functions.
  201: Test File Functions - 1: f_open, f_close, and f_unlink
  202: Test File Functions - 2: f_write and f_read
  203: Test File Functions - 3: f_lseek
  204: Test File Funttions - 4: f_stat, f_utime, f_chmod
  205: Test File Functions - 5: f_forward
  206: Test File Functions - 6: f_truncate
  207: Test File Functions - 7: f_sync
  208: Test File Functions - 8: f_printf, f_puts, f_putc, f_gets
  209: Test File Functions - 9: f_rename
  301: File/Dir: file operations on dirs & vice versa

Test case 101: Test Directory Functions - 1: f_mkdir, f_unlink functions with 0 of recursive level.
Test case passed

Test case 102: Test Directory Functions - 2: f_mkdir in cases of invalid directory names.
Test case passed

Test case 103: Test Directory Functions - 3: f_unlink in cases of invalid directory names.
Test case passed
Test case 104: Test Directory Functions - 4: f_mkdir, f_unlink functions with 1 of recursive level.
Test case passed

Test case 105: Test Directory Functions - 5: f_mkdir, f_unlink functions with 2 of recursive level.
Test case passed

Test case 106: Test Directory Functions - 6: f_chdir, f_getcwd, f_unlink functions.
Test case passed

Test case 107: Test Directory Functions - 7: Make maximum number of sub-directories.
Test case passed

Test case 108: Test Directory Functions - 8: f_opendir, f_readdir functions.
Test case passed

Test case 109: Test Directory Functions - 9: f_chdir function with .. directory.
Test case passed

Test case 110: Test Directory Functions - 10: f_readdir in case of there are many files in read
directory.
Test case passed

Test case 111: Test Directory Functions - 11: f_stat, f_utime, f_chmod functions.
Test case passed

Test case 201: Test File Functions - 1: f_open, f_close, and f_unlink
Test case passed

Test case 202: Test File Functions - 2: f_write and f_read
Test case passed

Test case 203: Test File Functions - 3: f_lseek
Test case passed
Test case 204: Test File Funttions - 4: f_stat, f_utime, f_chmod
Test case passed

Test case 205: Test File Functions - 5: f_forward
Test case passed

Test case 206: Test File Functions - 6: f_truncate
Test case passed

Test case 207: Test File Functions - 7: f_sync
Test case passed

Test case 208: Test File Functions - 8: f_printf, f_puts, f_putc, f_gets
Test case passed

Test case 209: Test File Functions - 9: f_rename
Test case passed

Test case 301: File/Dir: file operations on dirs & vice versa
Test case passed

Test cases:
Executed: 21, Passed: 21, Failed: 0
```

F.3.2.2 Test Group 2

This test group contains the following subgroup.

Table 5-5. Test Group 2

Subgroup	Description	FATFS module configuration
TestDir12	This test group is to test f_chdrive, f_getfree, f_mount functions. It also test multi-partition feature of FATFS module	#define _FS_TINY 1 #define _FS_READONLY 0 #define _FS_MINIMIZE 0 #define _MULTI_PARTITION 1 #define _VOLUMES 2 #define _FS_RPATH 2

To enable the test group, define the macro **RUN_TEST_112** in file **testcase.h**.

Expected results of the test case are shown in the HyperTerminal as follows.

```
FAT test
Waiting for USB mass storage to be attached...
Mass Storage Device Attached
Test Cases:
  112: Test Directory Functions - 12: f_chdrive, f_getfree, f_mount functions.

Test case 112: Test Directory Functions - 12: f_chdrive, f_getfree, f_mount functions.
Disk '0':
  ClusterSize: 4096
  TotalClusterCount: 1994
  TotalFreeClusterCount: 1994

Disk '1':
  ClusterSize: 512
  TotalClusterCount: 94874
  TotalFreeClusterCount: 94873

Test case passed
Test cases:
Executed: 1, Passed: 1, Failed: 0
```

F.3.2.3 Test Group 3

The test group consists of following subgroups.

Table 5-6. Test Group 3

Subgroup	Description	FATFS module configuration
TestFile10	This test group is to test file sharing policy.	#define _FS_TINY 1 #define _FS_READONLY 0 #define _FS_MINIMIZE 0 #define _MULTI_PARTITION 0 #define _VOLUMES 1 #define _FS_RPATH 0 #define _FS_SHARE 2 #define _USE_LFN 0
TestFile11	This test group is to test how FAT apis work when drive status is invalid.	
TestFile12	This test group is to test how FAT apis work when LFN is disable.	
TestFile13	This test group is to test how FAT apis work when _RS_PATH = 0.	

To enable the test group, define the macro **RUN_TEST_210_213** in file **testcase.h**.

Expected results of these test cases are shown in the HyperTerminal as follows.

```
FAT test
Waiting for USB mass storage to be attached...
Mass Storage Device Attached
Test Cases:
  210: Test File Functions - 10: file sharing policy
  211: Test File Functions - 11: invalid drive status - FR_NOT_ENABLED
  212: Test File Functions - 12: LFN disable
  213: Test File Functions - 13: _RS_PATH = 0

Test case 210: Test File Functions - 10: file sharing policy
Test case passed

Test case 211: Test File Functions - 11: invalid drive status - FR_NOT_ENABLED
Test case passed

Test case 212: Test File Functions - 12: LFN disable
Test case passed

Test case 213: Test File Functions - 13: _RS_PATH = 0
Test case passed

Test cases:
Executed: 4, Passed: 4, Failed: 0
```