Freescale MQX RTOS Example Guide
USB MSD Disk example

This document explains the msd_disk example, what to expect from the example and a brief introduction to the API.

The example

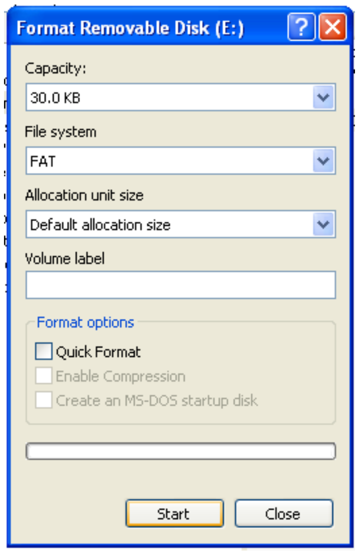
The msd_disk example enumerates as a USB Mass Storage Device. The demo uses one form of physical memory to present to the computer a memory drive. The USB stack enumerates in the computer host as a mass storage device with two bulk endpoints, one IN endpoint (device to PC) and one OUT endpoint (PC to device). Both endpoints are configured to use 64 byte buffers.

When the USB device enumerates, the host can access the device as a standard mass storage device, in most graphical operating systems this will show up as an accessible disk drive. The standard memory medium the demo uses is a section of RAM. Support for other storage media can be implemented by the user.

Running the example

Connect already running board to USB port of the host. Please do not confuse the MCU USB port cable and the BDM USB cable, MCU USB port has to be used in this case.

Once the USB cable is connected, the computer will enumerate the USB controller, this will usually cause an audible alarm and a pop-up to appear in the screen. The first time this is done, the computer should install drivers automatically as the mass storage device is a standard USB class. Once enumerated, the computer should show a new drive. Upon trying to open this drive, the computer should inform the user that the drive has no format and it needs to be formatted. Accept this prompt and a window similar to the next one should appear:



Format Removable Disk (E:)

Capacity: 30.0 KB

File system: FAT

Allocation unit size: Default allocation size

Volume label:

Format options
- [ ] Quick Format
- [ ] Enable Compression
- [ ] Create an MS-DOS startup disk

The disk will be formatted using FAT16 format which is standard type for a 30 KB memory (this is the size that the demo application prepares in the MCU or MPU RAM). The drive can be optionally named and then click on start.

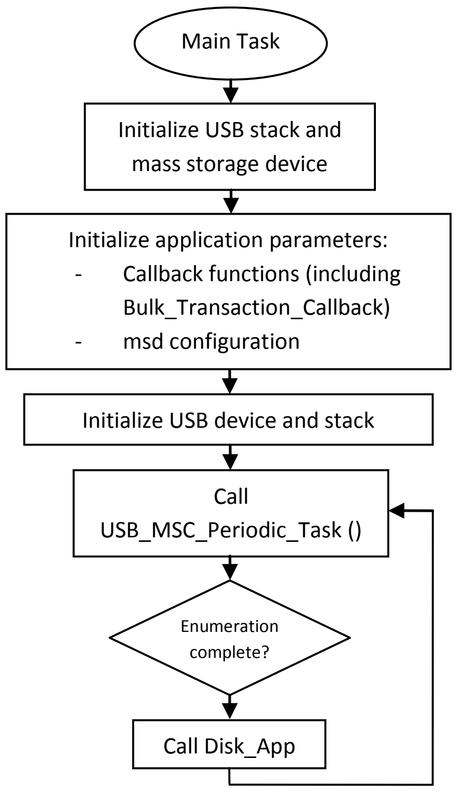
After the format process is done there should appear an empty disk. User can write and read files up to 30 KB in size to or from the drive.

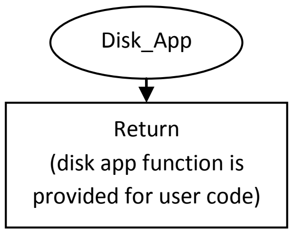
A basic test is to create a new text file and save it to the drive. Then disconnect the drive and reconnect it again to find that the file is still there. Please notice that this will only work if the board has an independent power supply (e.g. USB debug cable mentioned above), because the emulated drive is simply a section of RAM and if the board is powered only by the application USB cable it loses power when the cable is unplugged and the content of RAM is lost. When the board is powered down the expected behavior of the demo is that upon opening the drive once reconnected the PC host will request the drive to be formatted again.

This demo is basically just a building block for communicating to a USB host via exchanging files, many applications can be built this way, like a removable media reader (SD card, or other) or a firmware upgrade application that reads a file copied in the virtual drive and reprograms itself with it.

Explanation of the example

The application demo creates only one main task. The flow of the task is described in the next figure.


Main Task
Initialize USB stack and mass storage device
Initialize application parameters:
- Callback functions (including Bulk_Transaction_Callback)
- msd configuration
Initialize USB device and stack
Call USB_MSC_Periodic_Task ()
Enumeration complete?
Call Disk_App



The main task only calls the TestApp_Init function. The TestApp_Init function initializes all the application parameters. It first checks if the USB memory (for USB stack operation) is available and returns an error otherwise.

Aferward it initializes the USB callbacks. Most important here is the Bulk_Transaction_Callback which handles the memory operations. The Bulk_Transaction_Callback runs asynchronously together with the USB stack interrupts and will be explained below.

Afterwards the function passes the MSD configuration to the USB lower layers, which basically means copying user configurable values stored in macros in disk.h to the msd_config structure defined in usb_msc.h. These macros may be customized by user, however, deeper knowledge of USB protocol and the MQX USB API is recommended for this. The API documentation (MQXUSBDEVAPI) can be obtained from the Freescale website:

www.freescale.com/mqx

The TestApp_Ini finally initializes the stack and USB device and enters an infinite loop. The infinite loop calls two functions: USB_MSC_Periodic_Task and Disk_App.

USB_MSC_Periodic_Task is part of the USB stack and needs to be called periodically.

DiskApp is a function provided by the demo for users to write their own code that is dependent on USB execution. The function is only called if the device is properly enumerated, this is ensured by a check if the g_disk.start_app is TRUE.

The rest of the memory handling is done in the Bulk_Transaction_Callback function. This function is called by the USB stack after every mass storage class event and contains a switch-case decoder that executes functions according to the type of the event. The data received and data sent events are not handled by this demo, though user can write custom code on them. The most relevant events are USB_MSC_DEVICE_READ_REQUEST and USB_MSC_DEVICE_WRITE_REQUEST these call the memory handling functions that write and read the physical memory blocks. Notice that in these two handlers there is a conditional compiler statement that only compiles the RAM reading and writing code into the code if the RAM_DISK_APP macro is set, this allows for the user to disable the basic demo operation and include custom memory handling code instead. The other relevant event is USB_MSC_START_STOP_EJECT_MEDIA is command designed to start or stop physical media, which is in particular relevant for media with slow start-up like hard disks.