



Austin Design Center
11000 North Mopac Expressway
Stonelake Bldg. 6 Suite 500
Austin, Texas 78759

USB2230 Software Release Notes ***v0.436***

Updated 6-22-05

The information contained herein is confidential and proprietary to SMSC, shall be used solely in accordance with the agreement pursuant to which it is provided, and shall not be reproduced or disclosed to others without the prior written consent of SMSC. Although the information is believed to be accurate, no responsibility is assumed for inaccuracies. SMSC reserves the right to make changes to this document and to specifications and product descriptions at any time without notice. Neither the provision of this information nor the sale of the described semiconductor devices conveys any licenses under any patent rights or other intellectual property rights of SMSC or others. The product may contain design defects or errors known as anomalies, including but not necessarily limited to any which may be identified in this document, which may cause the product to deviate from published specifications. SMSC products are not designed, intended, authorized or warranted for use in any life support or other application where product failure could cause or contribute to personal injury or severe property damage. Any and all such uses without prior written approval of an officer of SMSC will be fully at the risk of the customer. SMSC is a registered trademark of Standard Microsystems Corporation ("SMSC").

SMSC DISCLAIMS AND EXCLUDES ANY AND ALL WARRANTIES, INCLUDING WITHOUT LIMITATION ANY AND ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND AGAINST INFRINGEMENT AND THE LIKE, AND ANY AND ALL WARRANTIES ARISING FROM ANY COURSE OF DEALING OR USAGE OF TRADE. IN NO EVENT SHALL SMSC BE LIABLE FOR ANY DIRECT, INCIDENTAL, INDIRECT, SPECIAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES; OR FOR LOST DATA, PROFITS, SAVINGS OR REVENUES OF ANY KIND; REGARDLESS OF THE FORM OF ACTION, WHETHER BASED ON CONTRACT; TORT; NEGLIGENCE OF SMSC OR OTHERS; STRICT LIABILITY; BREACH OF WARRANTY; OR OTHERWISE; WHETHER OR NOT ANY REMEDY OF BUYER IS HELD TO HAVE FAILED OF ITS ESSENTIAL PURPOSE, AND WHETHER OR NOT SMSC HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Software Compliance

The software in this release conforms to the following industry flash card specifications. SMSC has tested to the best of its ability to ensure that this software conforms to these specifications. However, no other warranty is assured, express or implied, other than provided by SMSC's standard terms and conditions.

1. SmartMedia™ Electrical Specification Version 1.40
2. SmartMedia™ Physical Format Specifications Version 1.40
3. SmartMedia™ Logical Format Specifications Version 1.30
4. MultiMediaCard System Specification Version 4.00
5. SD Memory Card Specifications Version 1.1
6. Memory Stick Standard Format Specification Version 1.41-00
7. Memory Stick Pro Standard Format Specifications Version 1.01-01
8. Memory Stick Duo Standard Format Specifications Version 1.10-00
9. CompactFlash Specification Rev 2.1
10. xD Picture Card Specification Version 1.2
11. Universal Serial Bus Specification Rev 2.0
12. USB Mass Storage Class, Bulk Only Transport Version 1.0

Table of Contents

Revision History	5
The Non-Volatile Store Data	6
Using Flash ROM to Store the NVStore Data	6
Creating the EEPROM.DAT File	6
Using the USB Drive Manager Application (for Windows XP only)	7
The Info Tab.....	8
The Branding Tab.....	8
Using .dat files with USBDM	8
The Configuration Tab	9
The Hub Tab.....	10
Attribute Bit Definitions	11
Setting the IrDA Transceiver Mode Pin Bit:	12
Setting MMC-4 Clock Speed and Card Power Management Bits:.....	13
Programming the NVStore Data.....	13
LUN Configuration and Icon Sharing.....	14
LUN Configuration	14
Icon Sharing	14
LUN Power Configuration	15
LUN Power Masks	16
Using Device Firmware Upgrade (DFU)	18
Overview	18
Files Required for DFU for Windows	18
Creating the 128KB DFU Capable Flash Binary “both.bin”	19
Preparing a Device for DFU Operation.....	20
Choosing a Flash Eeprom for Your Device.....	20
Setting up the Hardware	20
Using the USBDM Application to Perform Device Firmware Upgrade (DFU).....	21
Using the OEM.exe to Update Firmware	23
Creating a DFU Uploadable File	24
Using the DFU.exe Utility.....	25
Using the USB2230 Custom Icons Package.....	26
Contents of the USB2230 Custom Icons Package	26
Creating the Required SetIcon Ini Files.....	26
Manually Installing the Custom Icons Application Files	28
Troubleshooting the Custom Icons Application	30
Using the SMSC IrDA driver	31
Needed files for the USB2230 SMSC IrDA driver	31
Manually Installing the SMSC IrDA driver	33
Using the Automated Installer to Install the SMSC IrDA driver.....	33
Windows Installer Packages.....	33
Using the Production Line Descriptor Update Utility (PLDU)	34
Setting Up the PLDU Application.....	36
Using the PLDU to Update Device Descriptors	36
Using the Production Line Test Utility (PLTU).....	37
Creating the PLTU ini File	37
A Sample PLTU ini File	38
Setting Up the PLTU Application	39
Using the PLTU to Test Multiple Devices	39
Known Issues with the USB2230 Production Line Utilities.....	40
Using the QuickTest Production Line Read/Write Test Utility	41
Using the EPRMUPDT.exe Utility	42
Using the Windows XP Special Memory Stick Format Registry Key	44
Using the KillReg Utility	45
Using the Swapdrv Utility.....	46
Using the Dos Production Line Utility (DosPLTU)	47
Media Tested with the USB2230	51

USB2230 Performance Benchmarks 52

GPIO Assignment Table 53

Known Firmware Related Issues 54

 General: 54

 CF Devices: 54

 MS Devices: 54

 SM Devices: 54

 SD/MMC Devices: 54

 xD Devices: 54

Issues Not Related to Firmware 55

Revision History

0.390: **-ROM Mask 01.**

- Initial Release

0.409: **-External Evaluation Build.**

Firmware:

- Enabled firmware to detect a multiple-emulation error when the media reports a write failure in order to pass xD compliancy testing.
- Adheres to final 1.2 USB IrDA Bridge Protocol
- Fixed intermittent CRC errors found during FIR transfers
- Implemented SIP generation in order to guaranteed non-disruptive coexistence with slower systems (115.2kbit/s and below), once a higher speed (above 115.2 kbit/s) connection has been established.
- Fixed an issue with the IRDA connection disconnecting and reconnecting during media card transfers
- Fixed a bug where the MS media would not be recognized after a USB cable surprise removal was performed during a read from the MS.
- Changed iSerialNumber in the Device Descriptors to report 0x00 when the “Report the Serial number string as Zero” attribute bit is set.
- Added support for additional types of IrDA transceivers
- There is no support for device firmware update with this release of firmware. This functionality will be included in a future release.

Applications:

0.428: **-External Evaluation Build.**

Firmware:

- Added support for 4-bit High Speed MMC
- Added support for device firmware update
- Added support for IrDA transceiver shutdown during suspend
- Fixed a bug where the USB2230 device acting as a remote IrDA host fails to transfer single files whose sizes are greater than 450MB

Applications:

- USBDM (v2.0.0.3) Added dropdown option for MMC-4 attribute bits
- USBDM (v2.0.0.3) Added checkboxes for IrDA transceiver shut down bits
- USBDM (v2.0.0.3) Added dropdown option for CIR attribute bits(does not apply to 2230)
- USBDM (v2.0.0.3) Added capability to manually change attribute bits
- USBDM (v2.0.0.3) Reorganized Configuration tab for attribute bits to be separated by media type
- KillReg (v1.0.0.6) Added functionality to remove all registry entries associated with USB2230

0.436: **-ROM Mask 02.**

Firmware:

- Added IrDA MIR support.
- Changed Variable PID attribute bit and Idle Time Limit bytes to reserved

Applications:

- USBDM (v2.0.0.4) Removed Variable PID checkbox and Idle Time Limit field
- PLDU (v2.0.0.7) Removed Variable PID checkbox and Idle Time Limit field

The Non-Volatile Store Data

The NVStore is user modifiable data that is stored in either serial EEPROM or external program flash ROM and used by the device during operation. Some of the values that can be modified in the NVStore data include the serial number, VID/PID, Manufacturers ID String, Product ID String, LUN ID Strings, the modifiable device descriptors such as bmAttributes and MaxPower, number of LUNs, LUN order, and other modifiable bytes which customize the operation of the USB2230.

The NVStore data is programmed into the device using a text file “EEPROM.DAT”, which contains the bytes of data that are written to the EEPROM.

SMSC provides a utility to program the NVStore data called “USBDM.exe”. The procedure for using the USBDM Utility to write the NVStore data is described in the following paragraphs.

Using Flash ROM to Store the NVStore Data

If you are using external program flash you can, as a cost reduction measure, eliminate the need for a serial eeprom in your device by using the SST39VF010 Flash ROM, and the “NO EEPROM” version of the USB2230 firmware. The NO EEPROM firmware uses a portion of the memory storage area in the SST39VF010 Flash to hold all of the NVStore data. Currently, the SST39VF010 is the only chip supported by the NO EEPROM firmware. If you have a requirement to use another flash, please contact SMSC Sales to inquire about adding support for your chip.

Note: If external program flash is used, the access time of the flash media must be less than 66 nanoseconds.

Note: The USB2230 contains internal masked ROM program code. If you are running the 2230 from internal ROM code, you must use an external eeprom to store the NVStore data (VID/PID/Manufacturer and Product ID Strings, Attribute Bytes, etc.)

Creating the EEPROM.DAT File

An eeprom.dat file can be created using the USBDM application by altering all fields in the Configuration and Branding tab as desired for the new file and saving the file using the “Save” button. This can be done with or without a device attached to the host computer. The following section describes these tabs and the USBDM application in detail.

Using the USB Drive Manager Application (for Windows XP only)

The USB Drive Manager (USBDM) application can be used to create the eeprom.dat file and program the USB2230 device via USB, plus some additional functions such as creating end-user firmware updates contained within a single, easily distributable exe, and having the ability to instantly read the NVStore data from the device without the need for a driver swap.

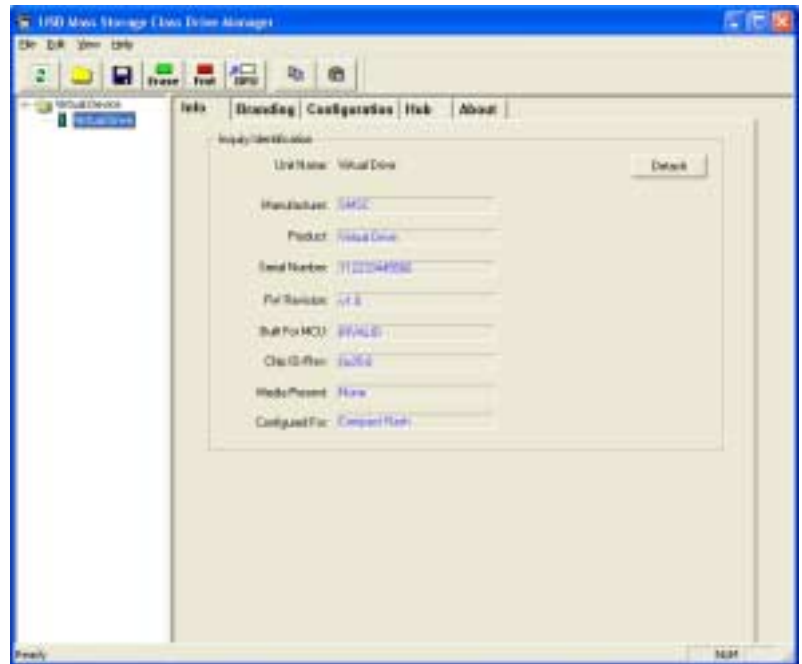
Note: Only USBDM version 2.0.0.4 or newer will work properly for updating 2230

Note: USBDM will not work for updating a SMSC standalone hub.

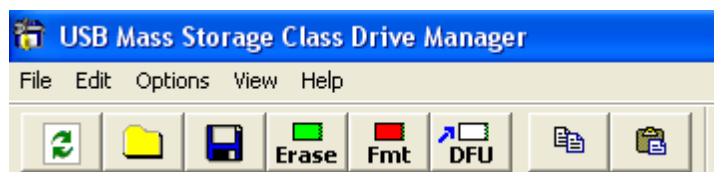
Note: The USBDM Application is supported in Windows XP only.

Getting Started:

To start the USB Drive Manager application, simply double click on the “USBDM.exe” executable. Once the application opens you will see the screen shown to the right if there is a device attached to the host computer. If there is no device present, a virtual device will be listed instead of the USB MSC Device information shown in this example. This virtual device allows a .dat file to be edited without the need for a device to be attached to the host computer.



The USBDM Toolbar



The toolbar buttons shown above are displayed at the top left hand side of the application. Starting from left to right, they perform the following functions:

- | | |
|--|---|
| Button 1: Refresh Drive List | Button 5: Format Drive (Not Used With 2230) |
| Button 2: Load .dat file | Button 6: Upload Firmware |
| Button 3: Save .dat file | Button 7: Copy |
| Button 4: Erase Media (Not Used With 2230) | Button 8: Paste |

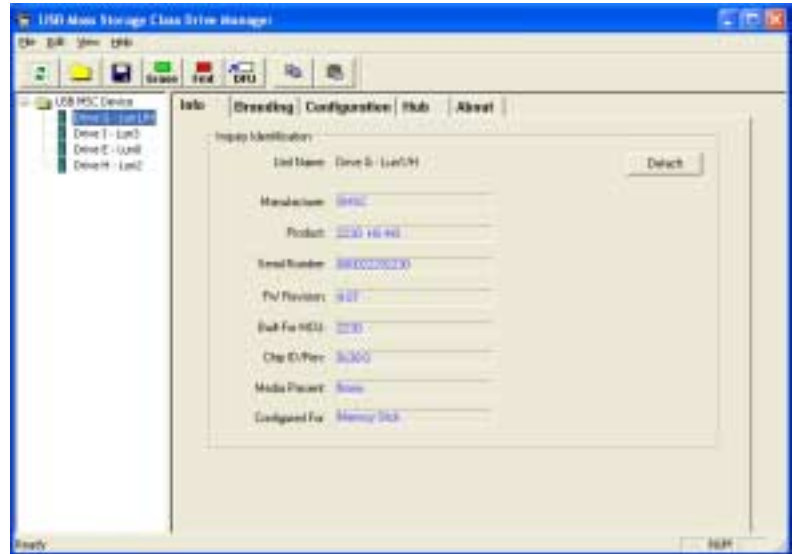
*If you do not see these buttons displayed, go to “View” in menu bar and make sure there is a check next to the “Toolbar” option.

*Clicking on the “Help” option above the toolbar and selecting “About Drive Manager” will display the version of the USBDM application.

The Info Tab

The info tab is displayed whenever a USB mass storage class device is attached to the host while USBDM is running. This tab displays the key fields in the NVStore data for the device. Note: Unless the device contains the SMSC USBDM firmware extensions, most of the data fields will display INVALID.

Attach a device containing the USBDM firmware extensions to the PC via a USB cable. The USB Drive Manager application will read the NVStore data for this device if there exists valid data. It will display information for each drive that is available on the device. The example to the right has information for Drive F, Drive G, Drive H, and Drive I. You can toggle between the information for each of these drives by single clicking on the Drive entry under the “USB MSC Device” folder on the left side of the application.



Note: The detach button seen on this tab will momentarily detach the target device from the system.

The Branding Tab

The Branding tab is used to write vendor specific data to the NVStore. Programmable fields include: Vendor ID, Product ID, Language ID, Product String, Manufacturing String, and Serial Number String. Any of this information can be changed on the device. Once you have entered the information for your device, click on the “Update Now” button to program the NVStore.

Vendor ID: Unique for every vendor. Assigned by the USB Implementers Forum.

Product ID: Unique to product. Assigned by vendor.

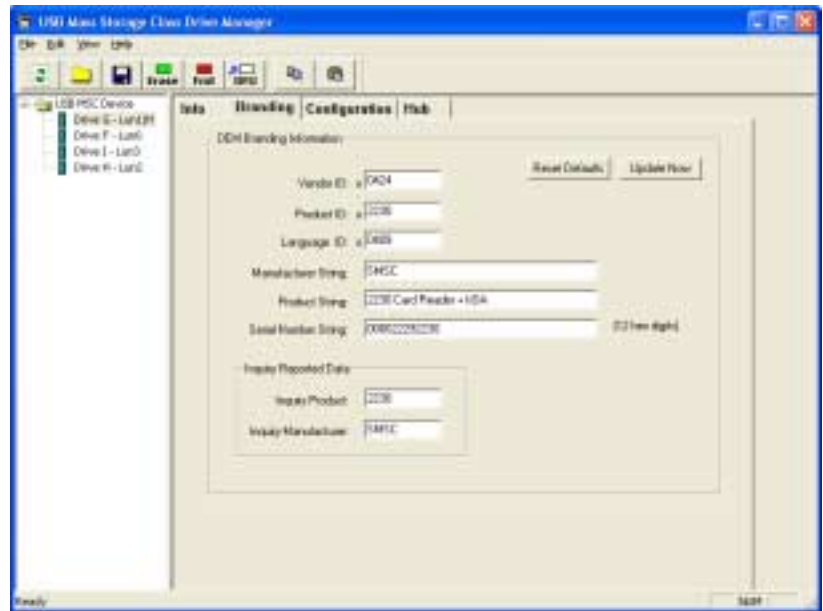
Language ID: 0409 is the Language Code for English. Other Language Codes may be found in the USB specification.

Product String: 28 characters max. Used to identify the product. This string will be used during the USB enumeration process in Windows.

Manufacturing String: 28 characters max. Used to identify the manufacturer.

Serial Number: 12 hex digits max. Must be unique to each device.

Inquiry Manufacturer (8 Bytes) and Product (5 Bytes) ID Strings: If bit 4 of the 1st attribute byte is set, the device will use these strings in response to a USB inquiry command, instead of the USB Descriptor Manufacturer and Product ID Strings.



Using .dat files with USBDM

The Load .dat file button can be used to populate these fields from a valid .dat file. After clicking the Load .dat file button, you will be prompted to specify a .dat file. Once the .dat file has loaded, the text fields will be updated to reflect the data in the .dat file. Any changes made to the text fields can also be saved into a .dat format using the Save .dat file button at the top of the application.

The Configuration Tab

The Configuration tab contains all of the other NVStore programmable fields not found in the Branding Tab.

The Configuration Tab is where you set:

- 1) The NVStore signature (always “ATA1” for USB2230)
- 2) The attribute bits
- 3) The LUN assignments
- 4) The LUN IDs
- 5) NAND Profile (Not Used for USB2230)
- 6) Miscellaneous settings such as the USB descriptors bMaxPower and bmAttribute

These user programmable fields are described in detail in the following paragraphs.

Signature: Signature should remain set to ATA1 for USB2230.

Attribute Bytes: This field should only be used in development phase to modify attribute bits that have not yet been implemented into check boxes and dropdown choices on this tab. There needs to be full understanding of what effect altering a specific bit will have on the device before changing this field. All released features will be able to be selected without utilizing this field.

Attribute Bits: The attribute bits are used to customize the functionality of the USB2230 firmware. They are organized by which particular media type they pertain to. Attribute bit checkboxes that are not specific to a media type are contained in the Misc. Settings section. A complete list of all programmable attribute bits and their function is listed in the section of this document entitled “Attribute Bit Definitions and NVStore Editable Values.” In the image shown above “Reverse SD Card Write Protect Sense” is the only option selected. Placing a check to the left of an option sets an attribute bit. If the box is unchecked, the attribute bit will be cleared. Any of these options may be checked or unchecked depending on the various needs for the product being programmed. There are also dropdown options in the “IrDA”, “CIR” and “MMC-4” sections. Only the “MMC-4” and “IrDA” dropdowns apply to the 2230 and are explained in detail in the Attribute Bit definitions section.

LUN Configuration:

LUN ID Strings (7 bytes each)- There are four LUN ID strings corresponding to LUN# 0,1,2 and 3.

Number of Icons to Display, CF Lun #, MS Lun #, NAND Lun #, SD/MMC Lun #, SM Lun #- These bytes are used to specify the number of LUNs the device exposes to the host. These bytes are also used for icon sharing- Assigning more than one LUN to a single icon. This is used in applications where the device utilizes a combo socket and the OEM wishes to have only a single icon displayed for one or more interfaces. For more information, see the section of this document entitled “LUN Configuration and Icon Sharing.” If this field is set to “FF”, the program assumes that you are using the default value of “04” and will display icons for CF, MS, SM, and SD. If this field is any other value besides “FF”, you must specify the LUN# assignments in the boxes starting with LUN 00 and going to (# of Icons to Display -1)

NAND Profile (2 Bytes): (Not used for the USB2230) This is where the NAND performance profile is specified for controllers that use it.

Note that more than one interface (CF, MS, SM, or SD) can share a LUN. Remember **LUN numbering always starts at 00**.

The configuration to the right directs the firmware to show three LUN’s in the order of CF, SD/MMC, and SM. Note that Memory Stick is not enabled in this configuration.

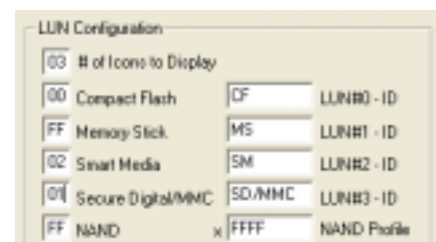
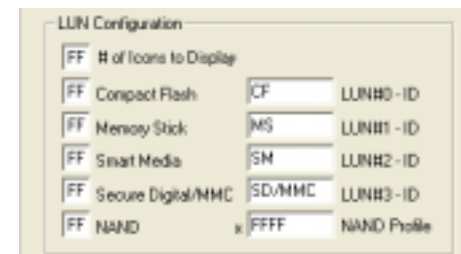
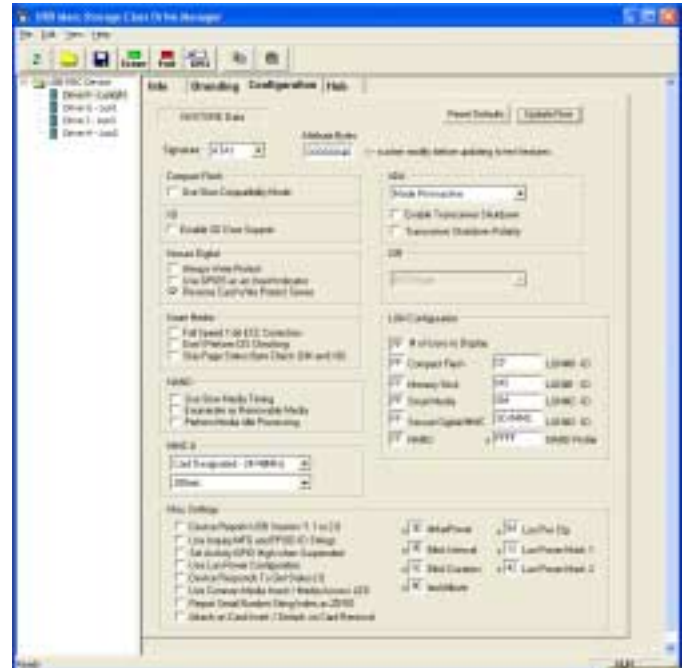
Of Icons to Display: 03

Compact Flash (1st LUN): 00

Memory Stick (will not display): FF

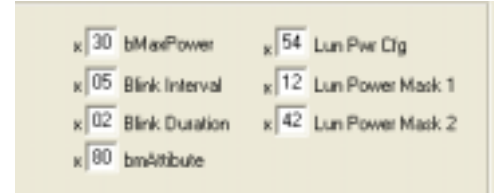
Smart Media (2nd LUN): 02

Secure Digital/MMC (3rd LUN):01



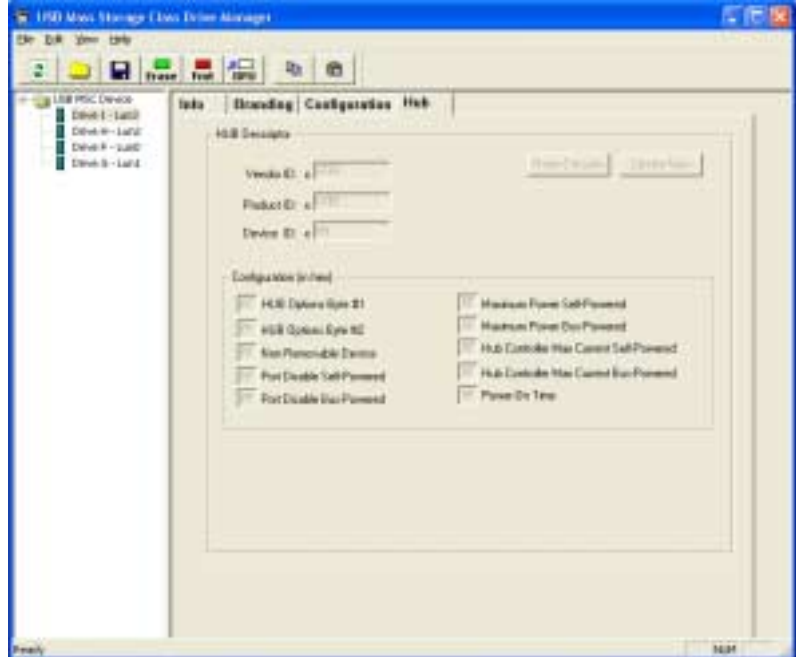
Misc. Settings: The Misc. Settings section is used to program the other miscellaneous NVStore editable values. They are:

- 1) **bMaxPower** (1 byte): Per USB specification. Do not set this value greater than 100mA
- 2) **Blink Interval** (1 byte): Programmable in 10ms intervals. Hi bit indicates idle state: 0–Off, 1–On. The remaining bits are used to determine the blink interval up to a max of 128 x 10 ms.
- 3) **Blink Duration** (1 byte): This byte is used to designate the number of seconds that the GPIO 0 LED will continue to blink after a drive access. Setting this byte to “05” will cause the GPIO 0 LED to blink for 5 seconds after a drive access.
- 4) **bmAttribute** (1 byte): Per USB Specification.
80 - Device is Bus Powered
C0 – Device is Self Powered
- 5) **Lun Pwr Cfg** (1 byte): – Should be a valid hexadecimal number. Default = 54. Refer to the “Lun Power Configuration” section for additional information on how to calculate this byte.
- 6) **Lun Power Mask 1** (1 byte): - contains the power mask setting for CF and MS controllers. The mask used depends on how the LUN is configured in the LUN Power Configuration byte. Refer to the “Lun Power Configuration” section for additional information on how to calculate this byte.
- 7) **Lun Power Mask 2** (1 byte): - contains the power mask setting for SM and SD controllers. The mask used depends on how the LUN is configured in the LUN Power Configuration byte.



The Hub Tab

The Hub tab is non-functional for the 2230 product. If a 2230 (or any USB mass storage class device that does not support this tab's functions) is connected to the host when USBDM is running, the entries will be grayed out and inactive. Attempting to modify the contents of the Hub tab will have no effect on the operation of the device. Changing any values on the Hub tab will have no effect, as these entries will be grayed out and inactive.



Attribute Bit Definitions

Attributes (4 bytes): The attribute value for your device is determined by the options selecting in the USBDM utility provided by SMSC. Changing the checkboxes and dropdown boxes and updating the device can update this information. These bits are defined below and organized by the Byte/Bit order. In the USBDM GUI, these bits are organized by which media type/feature they affect. The majority of these bits are displayed as checkboxes in the USBDM GUI. A few of them are displayed in dropdown options. The “CIR” dropdown option is not used with the 2230 and cannot be altered. The “MMC-4” and “IrDA” dropdowns do apply to the 2230 and are described in detail following the bit definitions. The bit definitions are as follows:

Note: The bit names are shown in bold below and correlate to how the attribute checkbox is labeled in USBDM. Not all checkboxes apply to the USB2230. Those bits that do not apply will specify, “Reserved – always set to 0” in the definitions below.

Byte 1, bit 0: Reserved – always set to 0. **Use Slow NAND FLASH Media Timing**

Byte 1, bit 1: Reserved – always set to 0. **Enumerate NAND Device as Removable**

Byte 1, bit 2: Reserved – always set to 0. **Use GPIO5 as an SD Card Insert Indicator**

Byte 1, bit 3: Report Serial Number String Index as ZERO

1 - Always report iSerial as zero in the device descriptor.

0(default) - Report non-zero iSerial in device descriptor if serial number is valid.

Byte 1, bit 4: Use the Inquiry Manufacturer and Product ID Strings

1 – Use the Inquiry Manufacturer and Product ID Strings.

0 (default) - Use the USB Descriptor Manufacturer and Product ID Strings.

Byte 1, bit 5: Set Activity GPIO High when Suspended.

1 – The activity LED GPIO is set to High when suspended.

0(default) - The activity LED GPIO is set to Low when suspended.

Byte 1, bit 6: Reverse SD Card Write Protect Sense

1 (default) - SD cards will be write protected when SW_nWP is high, and writable when SW_nWP is low

0 - SD cards will be write protected when SW_nWP is low, and writable when SW_nWP is high

Byte 1, bit 7: Make SD Cards Write Protected Always (Read Only)

1 – SD cards will always be write protected, regardless of the state of the card's write protect switch

0 (default) - SD cards will only be write protected when the write protect switch on the SD card is engaged

Byte 2, bit 0: Don't Perform Smart Media CIS Checking

1 – Ignore CIS check for Smart Media to allow the USB2230 to work with non-compliant cards.

0(default) – Enforce Strict CIS checking for Smart Media cards.

Byte 2, bit 1: Reserved – always set to 0. **Perform NAND Media Idle processing**

Byte 2, bit 2: Use Slow Compact Flash Compatibility Mode

1 – Compact Flash will operate in slow PIO-0 mode only regardless of CF card's actual capability.

0(default) – Compact Flash will operate at the fastest mode the card reports it can support.

Byte 2, bit 3: Device Responds To Get Status(1)

1 – Device will report itself as SELF POWERED in response to a GET STATUS from the host.

0(default) – Device will report itself as BUS POWERED in response to a GET STATUS from the host.

Byte 2, bit 4: Device Reports USB Version *1.1 or 2.0 (Warning: Setting this bit will result in the device being non-compliant with the USB 2.0 specification.)

1 – Device will report itself as USB version 1.1 in the bcdUSB device descriptor.

0(default) – Device will report itself as USB version 2.0 in the bcdUSB device descriptor.

Byte 2, bit 5: Use a Common Media Insert / Media Activity LED.

1 – The activity LED will function as a common media inserted/media access LED.

0(default) – The activity LED will remain in its idle state until media is accessed.

Byte 2, bit 6: Reserved – always set to 0. **Perform Software 1-bit ECC Error Correction on Smart Media.**

Byte 2, bit 7: Reserved – always set to 0. **Skip Page Status Byte Check on SM and xD.**

Attribute Bit Definitions (cont.)

Byte 3, bit 0: Reserved – always set to 0. **Attach on Card Insert / Detach on Card Removal.**

Byte 3, bit 1: Reserved – always set to 0. **Enable xD Door Support**

Byte 3, bit 2: Use Lun Power Configuration.

1 – Custom LUN Power Configuration stored in the NVSTORE is used. Refer to section “LUN Power Configuration” section for additional information about this feature.

0(default) – Default LUN Power Configuration is used.

Byte 3, bit 3: Reserved – always set to 0

Byte 3, bit 4: MMC-4 clock speed. (Set or cleared by dropdown option in MMC-4 section)

1 – Use 24 MHz clock only

0(default) – Use clock speed supported by card-24/48MHz.

Byte 3, bit 5: MMC-4 Card Power Management. (Set or cleared by dropdown option in MMC-4 section)

Combined with Byte 3, bit 6 to form the MMC-4 current allowed dropdown option. See section below titled “Setting MMC-4 Clock Speed and Card Power Management Bits” for additional information about this bit. This bit is set to 0 by default.

Byte 3, bit 6 MMC-4 Card Power Management. (Set or cleared by dropdown option in MMC-4 section)

Combined with Byte 3, bit 5 to form the MMC-4 current allowed dropdown option. See section below titled “Setting MMC-4 Clock Speed and Card Power Management Bits” for additional information about this bit. This bit is set to 0 by default.

Byte 4, bit 1: Use ATC programming versus dynamic control of the MODE pin (set or cleared by dropdown option in IrDA section. See section titled “Setting the IrDA Mode pin bit” below for additional information)

1 – Fast, Mode Pin Active, Use Dynamic ATC Programming

0 (default) –Slow, Mode Pin Inactive, Use Static Control of the Mode Pin.

Byte 4, bit 3: Enable Transceiver Shutdown

1 – Transceiver shutdown enabled

0 (default) – Transceiver shutdown disabled

Byte 4, bit 4: Transceiver Shutdown Polarity

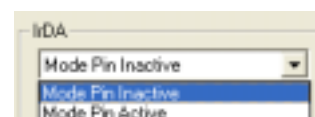
1 – Transceiver shutdown is active low

0 (default) – Transceiver shutdown is active high.

All other bits in the Attribute fields are reserved and should be set to 0.

Setting the IrDA Transceiver Mode Pin Bit:

The IrDA Transceiver Mode pin bit is Byte 4, bit 1. The description of this bit is listed in the previous section. This bit is set or cleared depending on what option is selected in the dropdown option in the IrDA section of the configuration tab in USBDM. The dropdown shown to the right, is used to set or clear Byte 3, bit 4. When “Mode Pin Inactive” is selected, Byte 4, bit 1 is set to 0. When “Mode Pin Active” is selected, Byte 4, bit 1 is set to 1.



When Enable Transceiver Shutdown is selected (Byte 4, bit 3 is set to 1) and ATC Programming is off (Byte 4 bit 1 is set to 0) then GPIO 12 is used to control transceiver shutdown.

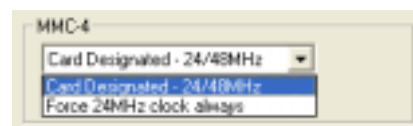
The following table lists the various transceivers that have been tested with the USB2230. It also lists the necessary setting for Byte 4, bit 1 for each transceiver.

Transceiver Manufacturer	Transceiver Part Number	Mode Pin selection (Byte 4, bit 1)	Transceiver Shutdown Pin
Vishay	TFDU6102	Mode Pin Inactive (0)	GPIO12
Agilent	HSDL-3602-007	Mode Pin Inactive (0)	GPIO12
Agilent	HSDL-3603-007	Mode Pin Active (1)	IRMode

Setting MMC-4 Clock Speed and Card Power Management Bits:

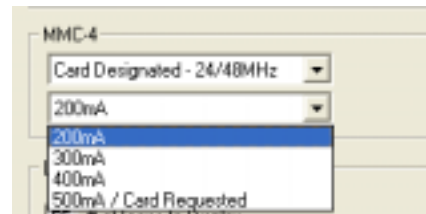
The MMC-4 Card Power Management bits are Byte 3, bit 5 and Byte 3, bit 6.

Most attribute bits are set by placing a check to the left of an option sets an attribute bit. This is not true for Byte 3, bit 4; Byte 3, bit 5; and Byte 3, bit 6. A dropdown box sets or clears these bits. Dropdown A shown to the right, is used to set or clear Byte 3, bit 4. When “Card Designated –24/48MHz” is selected, Byte 3, bit 4 is set to 0. When “Force 24MHz clock always” is selected, Byte 3, bit 4 is set to 1.



Dropdown A

Dropdown B shown to the right is used to set Byte 3, bit 5 and Byte 3, bit 6 by selecting the appropriate option in the dropdown box. The table below shows how each the dropdown choice correlates to the MMC-4 Card Power Management bits.



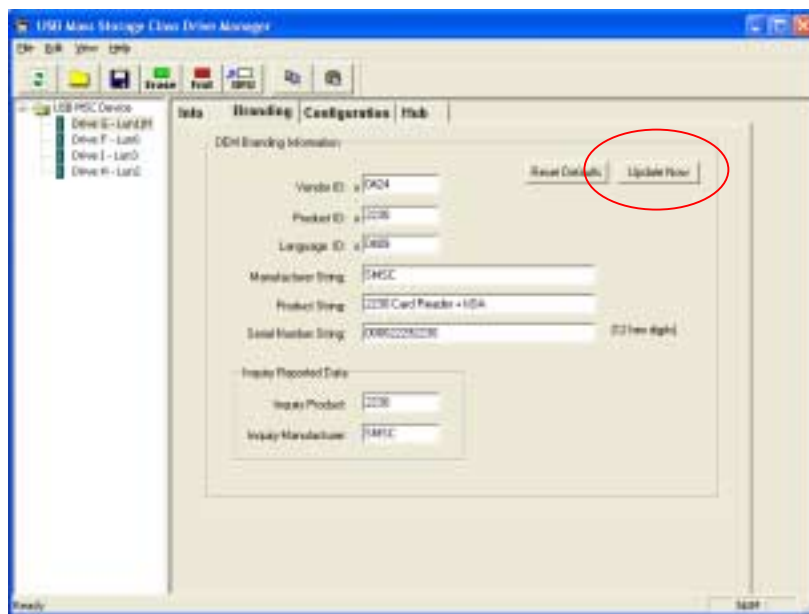
Dropdown B

Dropdown option (Current allowed)	Byte 3, bit 6	Byte 3, bit 5
200mA (default)	0	0
300mA	0	1
400mA	1	0
500mA or whatever the card requests	1	1

Programming the NVStore Data

Once the eeprom.dat file has been created and loaded into USBDM, you are ready to program the NVStore data into your device.

Press the “Update Now” button on either the Branding or Configuration Tab of the USBDM application. Both buttons will update all of the information displayed on any tab in USBDM. The operation will report that the Update completed Successfully once the data has been programmed.



LUN Configuration and Icon Sharing

LUN Configuration

LUN (Logical Unit Number) is the term given to each available media type in the USB2230. The USB2230 has a total of 4 LUNs available for use: Compact Flash, Memory Stick, Smart Media, and Secure Digital/Multimedia Card. OEMs can specify the number and order of LUNs exposed to the user by setting the LUN Configuration section of the Configuration tab in USBDM and updating the NVSTORE with these new settings.

Example: The example on the right shows the correct settings for a 2230 device that exposes icons for MS, SM and CF in that order. Note the following bytes:

Number of Icons to Display: “03” (The user will see 3 icons)
MS LUN #: “00” (Memory Stick will be the 1st icon displayed)
SM LUN #: “01” (Smart Media will be the 2nd icon displayed)
CF LUN #: “02” (Compact Flash will be the 3rd icon displayed)
SD/MMC LUN #: “FF” (An icon for SD/MMC will not be displayed)

Note: LUN numbering always starts at “00”.

LUN Configuration			
03	# of Icons to Display		
02	Compact Flash	CF	LUN#0 - ID
00	Memory Stick	MS	LUN#1 - ID
01	Smart Media	SM	LUN#2 - ID
FF	Secure Digital/MMC	SD/MMC	LUN#3 - ID
FF	NAND	x FFFF	NAND Profile

Icon Sharing

In addition to LUN configuration, the USB2230 can be further customized to allow more than one LUN to share an icon. This functionality would most likely be used for devices that contain multi-card adapters (adapters that can read more than one type of card.) So if you wanted to use a “5-in-1” or a “6-in-1” adapter, the USB2230 could be configured to only display a single icon to the user, rather than an icon for each individual media type. Alternatively, if you wanted to use a “4-in-1” adapter for Memory Stick, Smart Media, Secure Digital and Multimedia Card, but have a separate adapter for Compact Flash, you could configure the USB2230 to display 2 icons to the user (one for the 4-in-1 adapter and one for the Compact Flash) as shown in the example on the right.

Example: The example on the right shows the correct settings for a 2230 device that exposes 2 icons: 1 for (CF) and 1 for (MS, SM and SD/MMC) in that order. Note the following bytes:

Number of Icons to Display: “02” (The user will see 2 icons)
CF LUN #: “00” (Compact Flash will be the 1st icon displayed)
MS LUN #: “01”
SM LUN #: “01”
SD/MMC LUN #: “01” } (These media will all share a single icon)

LUN Configuration			
02	# of Icons to Display		
00	Compact Flash	CF	LUN#0 - ID
01	Memory Stick	MS	LUN#1 - ID
01	Smart Media	SM	LUN#2 - ID
01	Secure Digital/MMC	SD/MMC	LUN#3 - ID
FF	NAND	x FFFF	NAND Profile

LUN Power Configuration

The LUN Power Configuration allows the user to customize which GPIOs control power to which LUNs. Without this feature, users designing card readers that utilize multi-card sockets (sockets which can accept different flash card types) must include one FET for each card that the socket supports. Therefore, if a socket can accept any card type, the board design must include 4 FETs even though only 1 FET is active at a time. In order to reduce cost, only one FET is needed per socket. Users can set the LUN Power Configuration to have a single GPIO control power to the FET to deliver power to the multi-card socket, instead of requiring 4 GPIOs to power 4 FETs independently.

An additional feature for the 2230 is that it has 3 internal FETs which can be utilized instead of external FETs. The LUN Power Configuration feature allows any card (except CF cards) to be powered either by an external FET or internal FET. (CF cards can ONLY be powered by an external FET). Also, any card (except CF cards) can be powered by any combination of internal FETs. These features are configured via the NVSTORE settings. These configurations are described below.

In order to use this feature the user must set the “Use LUN Power Configuration” bit (Attribute byte 3 bit 2) and assign a valid hexadecimal number to the “LUN Pwr Cfg” byte (byte 172), “LUN Power Mask 1”, and “LUN Power Mask 2” in the NVSTORE.

The format of the NVStore LUN Pwr Cfg byte is as follows:

Bit							
7	6	5	4	3	2	1	0
SD Power GPIO	SM Power GPIO	MS Power GPIO	CF Power GPIO				

The Power GPIO field for each of the sockets shall be defined as follows:

Bit 1	Bit 0	Power GPIO
0	0	Use external FET, connected with GPIO 8,9,10 and/or 11
0	1	Use Internal FET, connected with GPIO 8, 10, or 11
1	0	Reserved
1	1	Reserved

By default the LUN Power Configuration byte will be as follows:

	LUN Power Configuration								Definition
	7	6	5	4	3	2	1	0	
CF							0	0	Use External FET
MS					0	1			Use Internal FET
SM			0	1					Use Internal FET
SD/MMC	0	1							Use Internal FET
	54h								

The above chart shows SD being powered by internal FET, SM powered by internal FET, MS powered by internal FET, and CF powered by external FET.

Note: When the appropriate attribute byte LUN Power GPIO is changed, the behavior of the Power GPIOs will change to that specified above regardless of LUN configuration.

LUN Power Masks

The LUN Power Masks are 4-bit fields that represent which GPIOs or FETs are configured for use with each LUN. The mask definition is different, depending on how the LUN is configured.

Power Mask Table

Config	Mask	FET(s)	PIN(s)
00	0001	External	GPIO 8
00	0010	External	GPIO 9
00	0100	External	GPIO 10
00	1000	External	GPIO 11
01	0001	Internal FET 0	GPIO 8
01	0010	Internal FET 1	GPIO 10
01	0011	Internal FET 0 and Internal FET 1	GPIO 8 and GPIO 10
01	0100	Internal FET 2	GPIO 11
01	0101	Internal FET 0 and Internal FET 2	GPIO 8 and GPIO 11
01	0110	Internal FET 1 and Internal FET 2	GPIO 10 and GPIO 11
01	0111	Internal FET 1 and Internal FET 2 and Internal FET 3	GPIO 8 and GPIO 10 and GPIO 11
01	1xxx	Invalid	Invalid

LUN Power Mask 1

LUN Power Mask 1 contains the power mask setting for CF and MS controllers. The mask used depends on how the LUN is configured in the LUN Power Configuration byte.

Bit							
7	6	5	4	3	2	1	0
MS Power Mask Default: 0001 – Internal FET 0				CF Power Mask Default: 0010 – External FET GPIO 9			

Default value for LUN Power Mask 1 is 0x12

LUN Power Mask 2

LUN Power Mask 2 contains the power mask setting for SM and SD controllers. The mask used depends on how the LUN is configured in the LUN Power Configuration byte.

Bit							
7	6	5	4	3	2	1	0
SD Power Mask Default: 0100 – Internal FET 2				SM Power Mask Default: 0010 – Internal FET 1			

Default value for LUN Power Mask 2 is 0x42

Example: The Icon Sharing example in the previous sections describes a device with 2 icons: 1 for (CF) and 1 for (MS, SM and SD/MMC) in that order. Since MS, SM and SD/MMC are all sharing a socket in that example only 2 FETs would be needed. The Lun Power Configuration feature can be used to assign two GPIOs to power these LUNs instead of the four GPIOs used by default. Suppose for example that the user would like the CF slot to be powered externally by GPIO 9 and the combo slot to be powered internally by FET0. First the user would set the attribute bit “Use LUN Pwr Config”. Then the user would set the LUN Power Configuration byte to 0x54, the LUN Power Mask 1 to 0x12, and the LUN Power Mask 2 to 0x11. (See tables below for how this value is found)

0x54 Example:

	LUN Power Configuration								Definition
	7	6	5	4	3	2	1	0	
CF							0	0	Use External FET
MS					0	1			Use Internal FET
SM			0	1					Use Internal FET
SD/MMC	0	1							Use Internal FET
	54h								

For MS, SM, and SD combo slot: From the Power Mask Table above we know that if the LUN Power Configuration for the media slot type is set to 0x01 and the desired power is from Internal FET 0, then the Power Mask for that media slot type is 0x0001.

For CF slot: From the Power Mask Table above we know that if the LUN Power Configuration for the media slot type is set to 0x00 and the desired power is from External GPIO9, then the Power Mask for that media slot type is 0x0010.

This information can be used to determine the LUN Power Mask bytes as follows:

Bit							
7	6	5	4	3	2	1	0
MS Power Mask				CF Power Mask			
0001 – Internal FET 0				0010 – External FET GPIO 9			
1				2			

LUN Power Mask 1 is 0x12

Bit							
7	6	5	4	3	2	1	0
SD Power Mask				SM Power Mask			
0001 – Internal FET 0				0001 – Internal FET 0			
1				1			

LUN Power Mask 2 is 0x11

Using Device Firmware Upgrade (DFU)

Overview

Device Firmware Upgrade (DFU) is the process by which device firmware is updated through a standard USB cable, eliminating the need to remove, reprogram and replace flash memory. This operation is accomplished by placing special code into an external flash memory chip at the time it is initially programmed. (Note: the external flash memory must have access time less than 66 nanoseconds in order for the firmware to run properly.) This code can then later be called upon to essentially change the USB device into a flash programmable device. Then new firmware can then be uploaded to the device and reprogrammed into the flash. Once the operation is complete, the device configures itself back to a normal USB device and begins utilizing the new firmware. **Please note that you can not perform a device firmware upgrade if you are running from the internal USB2230 ROM code. You must use an external flash with less than 66nS access time if you want to have device firmware upgrade capability.**

SMSC's Device Firmware Upgrade (DFU) package gives manufacturers the ability to easily utilize DFU to dynamically update the firmware and descriptor information in their devices. This allows for in circuit programming of new device firmware both on the assembly line, and by the end user in the field. This affords both the manufacturer and the end user a great opportunity to utilize the feature enhancements and bug fixes of new code immediately once it becomes available.

In order to help customers evaluate the DFU technology, SMSC provides a DFU package that consists of the DFU driver, device firmware, sample DFU applications and source code. This document serves to describe the use of these tools, and the implementation of Device Firmware Upgrade in a typical device application.

Files Required for DFU for Windows

USBDM.exe –A sample DFU application that demonstrates the procedure for updating the firmware and NVStore data.

eeprom.dat –A text file containing the changeable descriptor information used to update the NVStore. This file can be created and edited by changing the data in the Branding and Configuration tab in the USBDM application and saving the data.

hex2bin.exe -A batch capable utility that converts INTEL HEX, MOTOROLA 'S', or TEKTRONIX HEX files to Binary Format.

dfu.exe -A utility used to add, remove, or check for the presence of a DFU file suffix. Any firmware image that is to be uploaded to a device via DFU, should contain a valid DFU file suffix.

dfu2230.hex -The DFU execution code that is inserted into the lower 16kb of a 128kb flash when it is initially programmed. This hex file is merged with the 128K binary file "fmc.bin" with the "hex2bin.exe" utility to create the 128kb flash image. (Included with the USB2230 firmware).

fmc.hex -The USB2230 device firmware that is inserted into the upper 112kb of a 128kb flash when it is initially programmed. This hex file is converted to a 128kb binary file with the "hex2bin.exe" utility, and then merged with the 16kb "dfu.hex" file to create the 128kb flash image. (Included with the USB2230 firmware).

fmc.dfu -A firmware image that can be uploaded to the device. This file is created by the user. This document explains in detail how to make downloadable DFU images through the use of the "DFU.exe" utility, which appends a DFU file suffix to the firmware file to be uploaded to the device. (This file is created by the user).

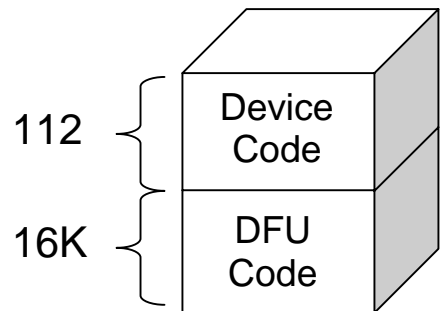
Application Source Code -All of the source code for the USBDM sample application.

Creating the 128KB DFU Capable Flash Binary “both.bin”

In order to prepare a device for DFU operation, the flash must be programmed with both the DFU code, and the normal USB2230 device code. The device code is converted to a 128KB binary file, and merged with the DFU code. Together they form the 128KB binary file which is uploaded to the flash eeprom. When this file is uploaded to the flash, the DFU code occupies the lower 16KB block, and the device code occupies the upper 112KB block.

In normal operation, a DFU capable USB2230 device executes only the device code in the upper 112KB block of memory. This code allows it to function as a normal USB 2.0 flash media controller. However, when the device is switched to DFU mode, the DFU code in the lower 16KB block begins executing and the device ceases to be a flash media device. Essentially, it changes to become an eeprom programming device. In this mode it is capable of reprogramming the USB2230 device code in the upper 112KB block of flash memory. Once the operation is complete, the device switches code execution back to the upper bank and begins operating with the newly updated code. At this point it ceases to be an eeprom programming device, and returns to being a flash media device.

128KB Flash EEPROM



To create the 128KB DFU capable flash binary file that will initially be programmed into the flash eeprom, you will need two files:

- 1) `fmc.hex` (The device code)
- 2) `dfu2230.hex` (The DFU code)

The “dfu2230.hex” file is provided by SMSC, and provides programming support for a limited number of eeproms. The “fmc.hex” file is the standard USB2230 device firmware. These two files, “dfu2230.hex” and “fmc.hex,” are both converted to binary files with the “hex2bin.exe” utility, and then merged with each other during the hex2bin command on the dfu file. Together they become the 128KB binary file “both.bin”. The procedure for creating “both.bin” is outlined below.

Note that this entire procedure can be accomplished easily using a simple DOS batch file:

```
hex2bin -l131070 fmc.hex fmc.bin  
hex2bin -m -l16384 dfu2230.hex fmc.bin
```

Preparing a Device for DFU Operation

In order to prepare a device for DFU operation, the flash must initially be programmed with the “both.bin” code. The “both.bin” file contains both the device code as well as the DFU code. The DFU code must preexist on the flash in order for it to be capable of receiving a DFU upload. The DFU code remains dormant in the lower 16KB of memory until it is called upon to perform a device firmware upgrade operation.

Note: The external flash used to program the “both.bin” must have access time less than 66 nanoseconds.

Once the flash has been programmed with the “both.bin” file, it may be inserted into the 2230’s flash socket in preparation for DFU operation.

Choosing a Flash Eeprom for Your Device

SMSC provides customers the “dfu.hex” file that supports only the SST39VF010, AM29LV010B, AM29LV040B, and the STM29W010B flash eeproms. While all of these flash support DFU firmware uploads, only the SST39VF010 supports NO EEPROM operation.

If you wish to use another flash in your device, it would most likely require some modification to the existing DFU code by SMSC to support the electrical characteristics of the new chip. If this is the case, please contact SMSC sales to have the project scheduled.

If you do decide to use another flash eeprom, there are a few requirements to look for to make sure it will work with DFU. First of all it should be 128KB and byte writable. It needs to have an access time of less than 66 nanoseconds in order for the external firmware to operate properly. Also, it should have equivalent programming characteristics as the supported chips, i.e. block size, erase size, read/write/erase speed, command set, and command address. Provided the chip meets all of the above requirements, there is a good chance that it will support DFU.

Setting up the Hardware

Either a USB 1.1 or 2.0 controller may be used for the DFU operation, however some USB 2.0 host controller drivers such as OMI’s have been found to have defects which prevent DFU from performing normally. If you are going to use a USB 2.0 host controller, it is recommended that you use Microsoft’s host controller drivers in order to achieve the best results. Once the board is attached and powered up, it should enumerate as a normal USB flash media controller. When you see the drive icon(s) appear, the device is ready. The following section describes the next step in the process, which is setting up the software application to perform the DFU.

Using the USBDM Application to Perform Device Firmware Upgrade (DFU)

The following files are needed to perform a device firmware upgrade with the USBDM application:

1. The USBDM application executable (USBDM.exe)
2. The device code (both.bin) ***Must be preprogrammed in the device flash in order to accept DFU**
3. A HEX to BIN converter (hex2bin.exe)
4. Utility to add the .dfu suffix (dfu.exe)
5. The updated firmware image. Steps to create this file are explained below (fmc.dfu)

* Note that if you also want to perform an update of the serial eeprom, you will need a 6th file, “eeprom.dat” which contains the descriptor information for the serial eeprom.

A firmware update can only be done using this application if a valid both.bin file is already programmed onto the device. See the section of this document entitled “Creating the 128KB DFU Capable Flash Binary ‘both.bin’” for steps on how to create the both.bin file.

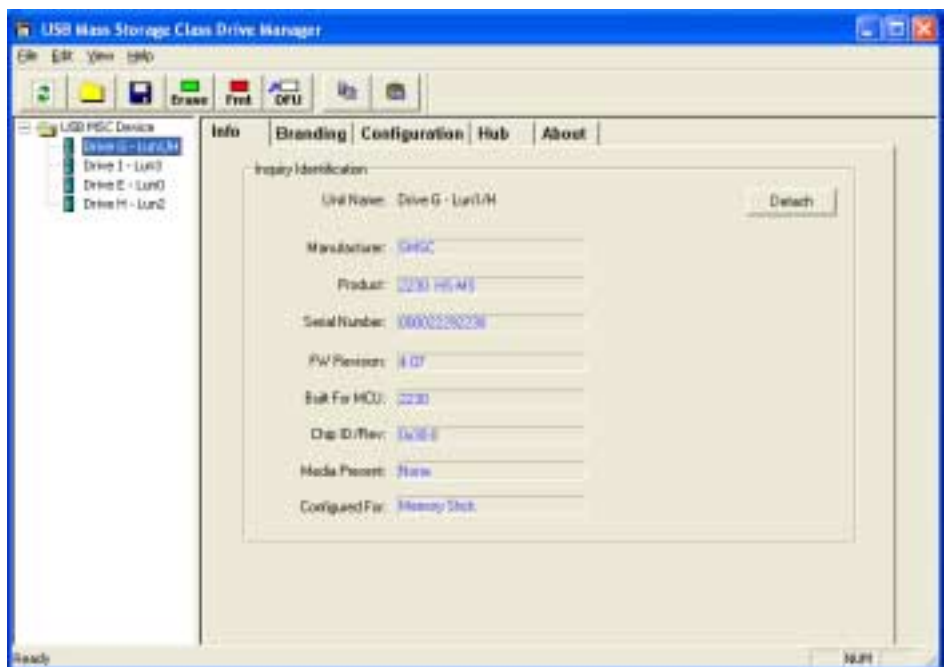
Creating the .dfu File:

The .dfu file is a DFU uploadable firmware image. It is essentially USB2230 firmware converted to binary format using the hex2bin.exe utility, with a DFU suffix appended to it. For information on creating the .dfu file, please see the section of this document entitled “Creating a DFU Uploadable File”. Please note that the USBDM application uses the device ID field (DID) to check firmware version information. The DID field should be filled with the major and minor firmware version (for this example, v4.28, the DID would be 0x0428).

This procedure can be completed using a simple DOS batch file:

```
hex2bin -l131070 fmc.hex fmc.bin
hex2bin -m -l16384 dfu2230.hex fmc.bin
dfu fmc.bin -did 0x0428 -pid 0x2230 -vid 0x0424
ren fmc.bin fmc.dfu
```

USBDM is used for the firmware update. To begin the firmware update, start USBDM by double clicking on the icon.



Updating the Firmware:

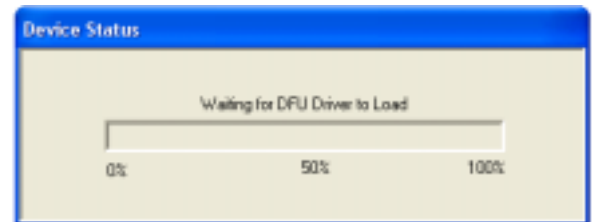
To perform a firmware update, click on the “Upload Firmware” button at the top of the application.



You will then be prompted to select the .dfu file that you wish to upload to your device. Navigate to the .dfu file (if it is not already listed in the current folder) and click open.



You will see a pop up box on your screen that displays the status of the firmware upload. This status will cycle through “Waiting for DFU Driver to Load”, “Switching to DFU Mode”, “Uploading New Firmware”, “Validating New Firmware”, and “Firmware Upload Successful”. Once the loading is complete you will be prompted to unplug the device and reattach it to continue (or to restart the host if the device is internally mounted). Once the device is reattached, the device will enumerate and the information for the updated firmware will be loaded into the USB Drive Manager application.



Note: The first time USBDM is used for DFU on a Windows XP host, the found new hardware wizard will be seen when the dfu driver is used during the firmware update process. This will only happen the first time a DFU is performed on a host. When this comes up, choose to have windows automatically install the driver. Choose to continue loading the SMSC DFU driver even though it is unsigned. While this is occurring, you may receive a message from USBDM asking you if you wish to continue waiting for the device to respond. Select yes to continue waiting.

Using USB Drive Manager to Create a Consumer Firmware Update Executable

USBDM can be used to create a very simple, easy to use, easy to distribute firmware update that OEMs can give to their customers to allow firmware upgrades. To create the executable, you only need two files:

1. The Drive Manager application (USBDM.exe)
2. The updated firmware image. (fmc.dfu)

Note: Ensure that the DID set in the DFU file matches the Major and Minor firmware revision.

Simply drag and drop the .dfu file on the USBDM.exe icon in Windows. You will see a popup box asking if you would like to create an OEM consumer version of the DFU application. Click yes and the application will build the consumer firmware update executable. The executable will be given the default name of “OEM.exe”. You can rename this file to whatever you like. This is the file that is distributed to the customer to allow firmware upgrades.



Note: The target device must be preprogrammed with a valid “both.bin” file to allow firmware upgrades.

Using the OEM.exe to Update Firmware

The OEM executable icon is shown to the right.



1) Double click on this executable to begin updating the firmware in your target device.

2) You will be prompted to attach a supported USB device.

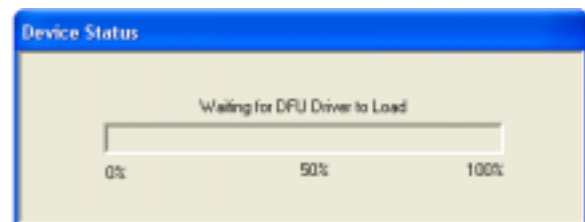
This prompt also displays which firmware version the executable will use to update your device. For this example, Firmware Version 3.00 is used.

3) Connect your device(if not connected already) and click “Continue”.



Note: This application allows consumers to make firmware updates to their device provided that 1) a valid both.bin file is already programmed on the target device and 2) the firmware that they are attempting to upgrade to is equal to or newer than the firmware version already on the device. This application will not allow an update to a version of firmware that is older than what is currently on the device. You will be asked if you would like to update your device firmware, click “yes” to verify the update and the application will begin to update your device.

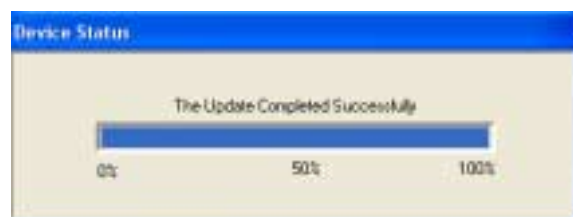
The application will show the status of the update. It will cycle through “Waiting for DFU Driver to Load”, “Switching to DFU Mode”, “Uploading New Firmware”, “Validating New Firmware”, and “Firmware Upload Successful”.



4) The USB Drive Manager application will prompt you to either reboot your computer (if an internal USB device was updated) or unplug the device and plug it back in (if an external device was updated).



After this is completed, you will see the device status pop up return with the message “The Update Completed Successfully”. The firmware is now updated on your device.

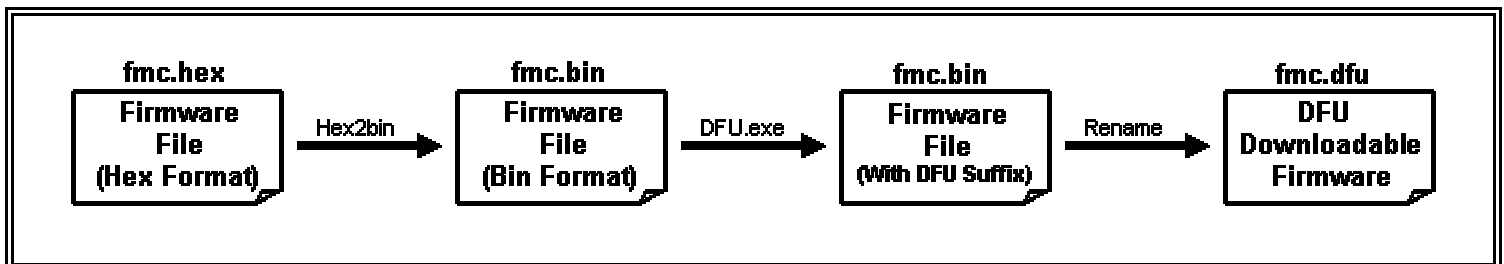


Creating a DFU Uploadable File

In order for a file to be uploadable via a DFU operation, it must contain a valid DFU file suffix. The DFU file suffix contains a CRC of the entire file, a DFU signature, and the VID, PID, and DID for the device to be upgraded. The following table was extracted from the USB Device Firmware Upgrade Specification (Rev 1.0), and shows the composition of the DFU file suffix.

Offset	Field	Size	Value	Description
-0	<i>dwCRC</i>	4	Number	The CRC of the entire file, excluding <i>dwCRC</i> . (Calculation specified in the following section).
-4	<i>bLength</i>	1	16	The length of this DFU suffix including <i>dwCRC</i> .
-5	<i>ucDfuSignature</i>	3	uc	The unique DFU signature field.
-8	<i>bcdDFU</i>	2	BCD	DFU specification number.
-10	<i>idVendor</i>	2	ID	The vendor ID associated with this file. Either FFFFh or must match device's vendor ID.
-12	<i>idProduct</i>	2	ID	The product ID associated with this file. Either FFFFh or must match device's product ID.
-14	<i>bcdDevice</i>	2	BCD	The release number of the device associated with this file. Either FFFFh or a BCD firmware release or version number.

In the SMSC DFU application, DFU downloadable files are given the extension “.dfu”. This is strictly arbitrary; the files can be of any extension as long as the application is designed to handle them. In order to create your own DFU downloadable file, you begin with the firmware file that is going to be used to upgrade the device. If the new firmware file is not already in binary format, it should be converted to binary using the Hex2Bin utility provided. Once in binary format, the “dfu.exe” utility is used to append a valid DFU file suffix to the firmware file (See the next section titled “Using the DFU.exe Utility”). Once the DFU file suffix has been added, you may rename the file with a .dfu extension to indicate that it is DFU downloadable. The entire procedure for creating the DFU downloadable file is summarized below.



Using the DFU.exe Utility

The “DFU.exe” utility can be used to add a DFU suffix to a file, or to check for the presence of a valid DFU suffix on an existing file. If required, the “DFU.exe” utility can also be used to remove a DFU suffix from a file. The “DFU.exe” utility is run from a command box in Windows.

The usage of DFU.exe is:

DFU.exe <filename> [options]

To check for the presence of a DFU file suffix:

DFU.exe <filename>

To remove a DFU suffix from a file:

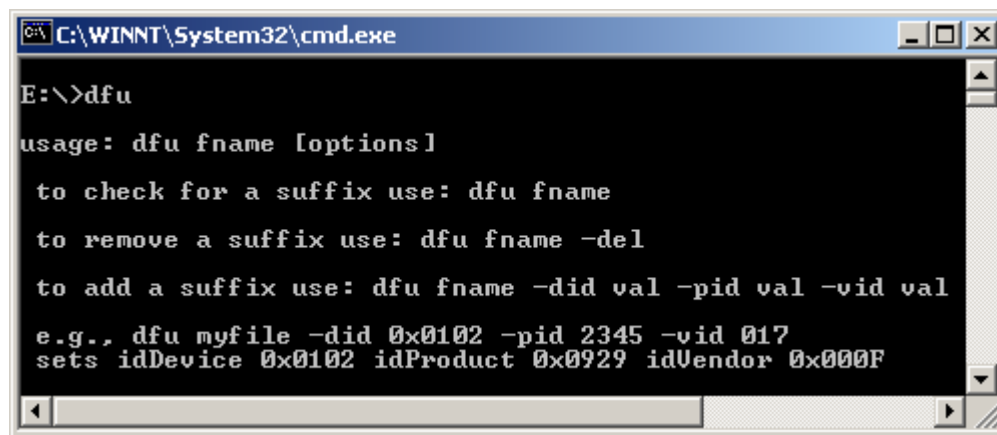
DFU.exe <filename> -del

To add a DFU suffix to a file:

DFU.exe <filename> -did <val> -pid <val> -vid <val>

Example of adding a DFU suffix to “fmc.bin”:

DFU.exe fmc.bin -did 0xFFFF -pid 0x2230 -vid 0x0424



```
C:\WINNT\System32\cmd.exe

E:\>dfu

usage: dfu fname [options]

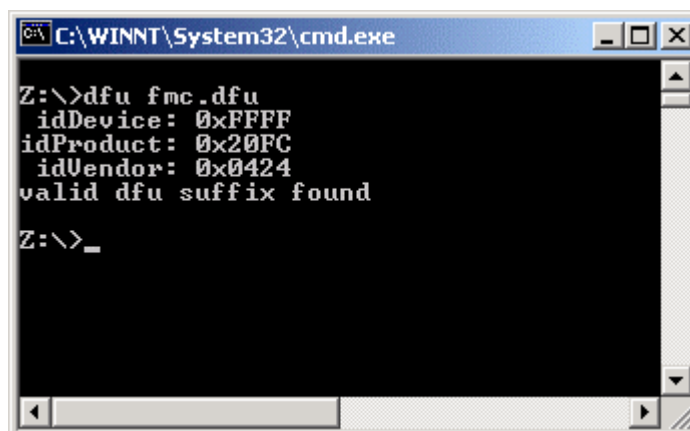
to check for a suffix use: dfu fname

to remove a suffix use: dfu fname -del

to add a suffix use: dfu fname -did val -pid val -vid val

e.g., dfu myfile -did 0x0102 -pid 2345 -vid 017
sets idDevice 0x0102 idProduct 0x0929 idVendor 0x000F
```

Once the DFU suffix has been added to the file, the last step is to give it a file extension that matches the type expected by your application. The dfuTest sample application is programmed to accept DFU uploadable files that have the “.dfu” extension. Finally, to check and make sure that the file has a valid suffix:



```
C:\WINNT\System32\cmd.exe

Z:\>dfu fmc.dfu
idDevice: 0xFFFF
idProduct: 0x20FC
idVendor: 0x0424
valid dfu suffix found

Z:\>_
```

Using the USB2230 Custom Icons Package

The USB2230 custom icons package allows OEMs to assign custom icons to the drives associated with the USB2230 flash media controller. This allows the end user to easily distinguish between the different media types in Windows Explorer. A new feature available in SetIcon is the ability to dynamically change icons based on media state. In other words, you can specify that one icon appear if there is media in the reader slot, and another icon appear when there is no media in the reader slot. Also, the dynamic icon functionality enables the detection of MMC, MS Pro, and xD, allowing the user to display custom icons for those media types as well.

Contents of the USB2230 Custom Icons Package

The USB2230 Custom Icons Package consists of the following:

SetIcon.exe- The custom icon application.

Smsc.ini- A sample Windows ini file.

Sample Icons- The sample icons distributed with this package are for evaluation use only.

Eeprom.dat- A text file containing the changeable descriptor information used to update the serial eeprom with the USBDM utility.

Creating the Required SetIcon Ini Files

In order for the SetIcon application to work properly, an ini file with a specific file name and format must be installed on the host computer. The ini file tells the SetIcon application which icons are associated with which drives, and provides a full path to each icon. The following four paragraphs describe the procedure for creating, naming, formatting and installing the ini file on the host PC.

1) Setting the Ini File Name:

Windows XP - The name of the ini file should be the same as the device's Manufacturer string, but be no longer than 8 characters. If the Manufacturer string is greater than 8 characters, then only the first 8 characters of the string should be used. If the Manufacturer string is less than 8 characters, then the ini file should use the entire Manufacturer's string.

Example: If MFG string is "Standard Microsystems Corp", the ini filename should be "Standard.ini"

Example: If MFG string is "SMSC", the ini filename should be "SMSC.ini"

(Note: The Manufacturer's string may be set or viewed using the USBDM Branding Tab. See the "Using the USB Drive Manager Application" section of this document for more details.)

Creating the Required SetIcon Ini Files (Cont.)

2) Setting the Ini Section Name:

Windows XP - The name of the section should be same as the first 5 characters of the Device's Product ID string enclosed in square brackets, including any spaces if present.

Example: If the Product ID string is "223 USB Controller", the section name should be "[223 U]"

Example: If the Product ID string is "223US", the section name should be "[223US]"

Example: If the Product ID string is "223", the section name should be "[223]"

Example: If the Product ID string is "", the section name should be "[]"

(Note: The Manufacturer's string may be set or viewed using the USBDM utility Branding Tab.)

3) Creating the Ini Section Content:

Under the Ini Section name should be a two line entry for each media type. The format for the two line entry is "Prod=Path\IconName.ico", where "Prod" is the string following the dash (-) in the Disk Drives section of the Device Manager for that drive (as seen in the screenshot to the right). Path\IconName.ico is the full path and icon name for the icon to be used for that drive. "ProdLABEL=Label Name" – (A declaration used to display a descriptive label in Windows Explorer for disk volumes with no names) where "ProdLABEL" is the same as "Prod" as explained above appended with the word "LABEL" and "Label Name" is the label that is to be displayed for the corresponding drive.

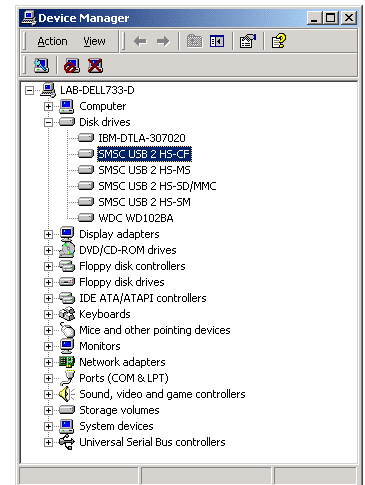
Note: The string length of "Label Name" should be less than 32 characters and should only contain alpha-numerical characters and special characters 'space' (' ') and 'under score' ('_').

Example: CF=\Program Files\Icons\CF.ico

Example: CFLABEL=Compact Flash Drive

Example: SD/MMC=\Program Files\Icons\SDMMC.ico

Example: SD/MMCLABEL=SDMMC Drive (Note there is no slash “/”)



Important Notes:

- 1) The full path to the icon should be less than 64 characters.
- 2) The file containing the icon should only be an .ico, .dll or .exe file.
- 3) There should not be any extra spaces before and after the '=' sign

To use the dynamic icon functionality, you also need to add lines for each LUN number and interface type (i.e. CF, SM, XD, etc.) for both the media present “L#_” and media not present “L#_NM” states. Please see the sample ini file that follows for clarification.

4) Placing the Ini File in the Correct Location on the Target PC:

In order for the custom icon application to work correctly, the ini file must be placed in one of the Windows System directories, depending on which operating system is being used. Those directories are:

Windows XP - "Windows\System32"

Manually Installing the Custom Icons Application Files

In order to perform a manual installation of the custom icons application files, the following steps should be performed:

1. Copy the SetIcon.exe file to a location on the target computer's hard drive. (i.e. "C:\Program Files\Icons\SetIcon.exe")
2. Copy the icon files to a location on the target computer's hard drive. (i.e. "C:\Program Files\Icons\").
3. Add a String entry to the Windows registry key "HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run" that will automatically start the SetIcon application each time the host computer is booted.

String: SetIcon **Value:** C:\Program Files\Icons\SetIcon.exe

4. Copy the ini file to the appropriate Windows System directory on the host PC. (See the previous section "Creating the Ini Files" for details.)
5. Manually start the SetIcon.exe application by double clicking it, or simply reboot the host PC. The entry placed in the registry during Step 3 will automatically start the application after the PC is rebooted.

A Sample Ini File

```
[2230 ]
CF=C:\Program Files\Icons\CF.ico
CFLABEL=Compact Flash Drive
MS=C:\Program Files\Icons\MS.ico
MSLABEL=Memory Stick Drive
SM=C:\Program Files\Icons\SM.ico
SMLABEL=Smart Media Drive
SD/MMC=C:\Program Files\Icons\SDMMC.ico
SD/MMCLABEL=SDMMC Drive

L0_CF=\Program Files\SMSC\Cf.ico
L0_CFLABEL=Compact Flash Drive
L0_NM=\Program Files\SMSC\cf-gray.ico
L0_NMLABEL=Compact Flash Drive

L1_MS=\Program Files\SMSC\Ms.ico
L1_MSLABEL=Memory Stick Drive
L1_MSPR=\Program Files\SMSC\MsPro.ico
L1_MSPRLABEL=Memory Stick Pro Drive
L1_NM=\Program Files\SMSC\ms-gray.ico
L1_NMLABEL=Memory Stick Drive

L2_SM=\Program Files\SMSC\Sm.ico
L2_SMLABEL=Smart Media Drive
L2_XD=\Program Files\SMSC\Xd.ico
L2_XDLABEL=xD Media Drive
L2_NM=\Program Files\SMSC\sm-gray.ico
L2_NMLABEL=Smart Media Drive

L3_SD=\Program Files\SMSC\Sd.ico
L3_SDLABEL=SD Media Drive
L3_MMC=\Program Files\SMSC\Mmc.ico
L3_MMCLABEL=MMC Media Drive
L3_NM=\Program Files\SMSC\sdmmc-gray.ico
L3_NMLABEL=SDMMC Media Drive
```

Creating a Windows Installer for the Custom Icons Application Files

Using an automated installer is the preferred method for installing and setting up the Custom Icons application to run on an end user's PC. As part of the USB2230 Custom Icons Application Package, a sample Windows installer is included which demonstrates a practical example of using a Windows installer to install, setup and run the Custom Icons application. To use the installer, simply run it and then reboot the host PC once the installation is complete. When the reboot is complete, the custom icons for the 2230 should appear in Windows Explorer.

Important Note: The ini files that are installed by the SMSC provided installer are hard coded to match SMSC's VID/PID, Manufacturer String, and Product ID String. The EEPROM.DAT file that is included with the software distribution contains the required data, and should be used to program evaluation boards to be used with the installer. Otherwise the ini files will not match the data in your board, and the icons will not appear. In general, to create a Windows Installer you should configure it to do the following:

1. Copy the SetIcon.exe file to a location on the target computer's hard drive. (i.e. "C:\Program Files\Icons\SetIcon.exe")
2. Copy the icon files to a location on the target computer's hard drive. (i.e. "C:\Program Files\Icons\").
3. Add a String entry to the Windows registry key "HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run" that will automatically start the SetIcon application each time the host computer is booted.

String: SetIcon **Value:** C:\Program Files\Icons\SetIcon.exe

4. Configure the installer to do a conditional installation depending on the operating system, to copy the ini files to the appropriate Windows System directory. (See the section "Creating the Ini Files" for details.)
5. Configure the installer to run the "SetIcon.exe" application once the install is complete. Alternatively, you could force the user to reboot the PC.

Troubleshooting the Custom Icons Application

Issue:

Cause:

After installing the Custom Icons application and rebooting, the custom icons do not appear.	<ol style="list-style-type: none"> 1) If you used the custom installer it is likely that the contents of your serial eeprom do not match the ini files that are installed with the installer. Read the section "Programming the Serial EEPROM" and program the eeprom to match SMSC's VID/PID, Manufacturers String, and Product ID String for the 2230. An EEPROM.DAT file with this data is included in the SetIcon software release for your convenience. 2) If you created your own ini files and installed the application files manually, the cause is most likely an incorrectly named or formatted ini file. Refer to the section "Creating the Ini Files" and double check to make sure that the ini files are correctly named, formatted, and placed in the proper location. 3) Check to see that the "SetIcon.exe" application is running by checking the Processes tab in the Task Manager.
After installing the Custom Icons application the drives still show the original icon.	Unplug the USB cable and then reattach it. Icons are only displayed when the device is attached with the SetIcon application running. If this does not correct the problem, try the troubleshooting steps above.
In Windows XP (SP1) the custom icons do not appear after a reboot of the host. However if the USB cable is detached and reattached, or media is either inserted or ejected, the icon(s) appear.	This is a bug in Windows XP. Microsoft has developed a fix (KB823293).
In Windows XP, the drive media label is not updated when a card is inserted.	This is a known issue in Windows XP. As a workaround, you can either hit F5 to refresh the label, or remove and reinsert the media.
Card Reader Software Installer does not install properly when attempting to run the installer after removing a previous version of the installer	When the Card Reader Software Installer is removed, the uninstall stops the device and removes all registry entries associated with the device. The device must be unplugged and reattached before it will enumerate.

Using the SMSC IrDA driver

The USB2230 SMSC IrDA driver needs to be properly loaded in order for IrDA to function with the USB2230. This driver only works with Windows XP. The driver may be installed manually or automatically by running the installer provided. The process for both methods is detailed below.

Needed files for the USB2230 SMSC IrDA driver

The following files are needed in order to load the SMSC IrDA driver

- 1) smscuir.sys
- 2) smscuir.inf

Below is a sample of the smscuir.inf file

```
; Smscui.inf
; Copyright 2004-2005, SMSC

[Version]
Signature = "$Windows NT$"
Class = Infrared
provider = %SMSC%
ClassGUID = {6bdd1fc5-810f-11d0-BEC7-08002BE2092F}
DriverVer = 11/30/2004,1.3.0.7
CatalogFile=SmscUIR.cat

[Manufacturer]
%SMSC%=SMSC

[ControlFlags]
ExcludeFromSelect = *

[SMSC]
%USB\VID_0424&PID_2230&Mi_01.DeviceDesc%=SmscUIR.Dev,USB\VID_0424&PID_2230&Mi_01

[DestinationDirs]
SmscUIR.CopyFiles = 12
DefaultDestDir = 12

[SmscUIR.Dev.NT]
CopyFiles=SmscUIR.CopyFiles
AddReg=SmscUIR.AddReg, SmscUIR.Params.AddReg
BusType=15
Characteristics = 0x04; NCF_PHYSICAL

[SmscUIR.Dev.NT.Services]
AddService = SmscUIR, 0x00000002, SmscUIR.AddService, common.EventLog

[SmscUIR.AddService]
DisplayName = %SmscUIR.SvcDesc%
ServiceType = 1 ; SERVICE_KERNEL_DRIVER
StartType = 3 ; SERVICE_DEMAND_START
ErrorControl = 1 ; SERVICE_ERROR_NORMAL
ServiceBinary = %12%\Smscui.sys
LoadOrderGroup = NDIS
```

[SmscUIR.AddReg]

HKR, Ndi, HelpText, 0, %SmscUIR.Help%
HKR, Ndi, Service, 0, "SmscUIR"
HKR, Ndi\Interfaces, UpperRange, 0, "ndisirda"
HKR, Ndi\Interfaces, LowerRange, 0, "nolower"
HKR, Ndi, RequiredAll, 0, "MS_IrDA"

[SmscUIR.Params.AddReg]

; Nothing as of now

[common.EventLog]

AddReg = common.AddEventLog.reg

[common.AddEventLog.reg]

HKR, , EventMessageFile, 0x00020000, "%SystemRoot%\System32\netevent.dll"
HKR, , TypesSupported, 0x00010001, 7

[SmscUIR.CopyFiles]

Smscui.sys

[SourceDisksFiles]

Smscui.sys=1

[SourceDisksNames]

1 = %DRIVERDISK%,,,

;-----;

[Strings]

SMSC="SMSC"

USB\VID_0424&PID_2230&Mi_01.DeviceDesc="SMSC USB CardReader-IrDA Adapter"

SmscUIR.SvcDesc="Smscui.sys SMSC USB CardReader-IrDA Adapter"

SmscUIR.Help = "The USB-IrDA bridge can be used to establish wireless serial and network links with other IrDA devices."

DRIVERDISK=""

Manually Installing the SMSC IrDA driver

In order to perform a manual installation of the SMSC IrDA driver, the following steps should be performed:

1. Copy the smscuis.sys file to the “Windows\System32\drivers” directory.
2. Copy the smscuis.inf file to the “Windows\inf” directory.
3. Attach a USB2230 device. The “Found new Hardware” wizard will come up.
4. Click next to have Windows XP search for the driver to use.
5. After several moments, Windows will find the SMSC IrDA driver and display a message asking the user if they want to continue loading the SMSC USB CardReader-IrDA Adapter driver even though it is unsigned. Click “Continue Anyway” to finish the install

Using the Automated Installer to Install the SMSC IrDA driver

In order to perform an automated installation of the SMSC IrDA driver, the following steps should be performed:

1. Attach a USB2230 device. Double click on the “Card Reader - IrDA Software Installer vXXX.exe”.
2. Continue to click “Next” through the installation process. Once the installation completed, the SMSC IrDA driver will be loaded properly without any user intervention.

Windows Installer Packages

The “Card Reader - IrDA Software Installer vXXX.exe” is a sample installer included in the USB2230 DFU and Driver Package and USB2230 Eval Board Package. This installer demonstrates a practical example of using a Windows installer to 1) install, setup and run the Custom Icons application and 2) install the SMSC IrDA Driver. To use the installer, simply run it and then reboot the host PC once the installation is complete.

Using the Production Line Descriptor Update Utility (PLDU)

Purpose: The PLDU is used to update device firmware and/or device descriptors such as the VID/PID, Manufacturer and Product ID strings in a production line environment using Windows 2000 SP3 and SP4. Under Windows XP, this can be used to update device descriptors or firmware if all the devices have same descriptor data. Otherwise, each device will enumerate as a MSC device and the utility needs to keep swapping drivers which is a time consuming operation and not really effective under a production line environment. This application is intended to be used by OEMs in their production line environment and is not intended for other users. The utility features a simple interface that displays success or failure of the programming operation in graphical form using either a green box with a checkmark (PASS), or a red box with an "X" (FAIL). The PLDU is capable of programming one device at a time and takes approximately 12 seconds to complete. The IRDA functionality of 2230 is currently only supported in Windows XP, however; this application can still be used in Windows 2000 to update the Mass Storage portion of the 2230.

Features:

1. Firmware update.
2. Descriptor (NVRAM) update.
3. Read descriptor (NVRAM) data from device.
4. GUI editor to edit and create DAT files.
5. Graphical and Text status display.
6. Automatic serial number increment after every descriptor update.
7. Break up of serial number to YY-MM-DD-S-SN format where
 - YY - Year (2 digits)
 - MM - Month (2 digits)
 - DD - Day (2 digits)
 - S - Station number (1 digit)
 - SN - Serial number (5 digits)

Application Behavior:

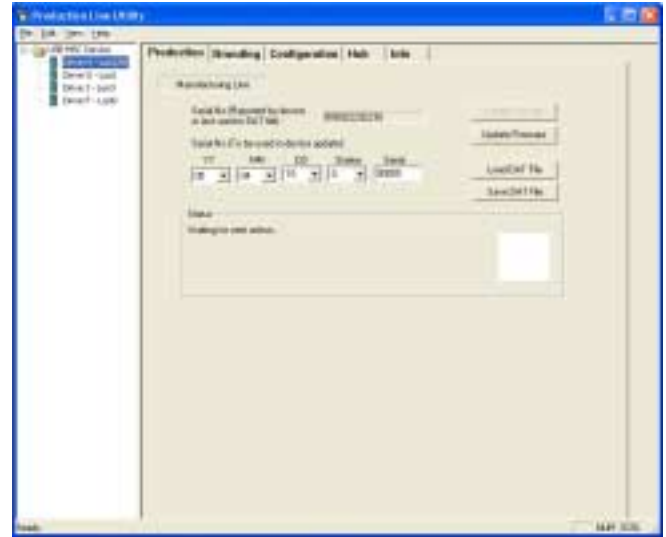
1. When the application is run with no SMSC devices plugged in, all controls should be disabled.
2. While the application is running, the controls should be dynamically enabled and disabled as the device is plugged and unplugged.
3. The button controls on all tabs except the "Production" tab will always be disabled.
4. The "Update Device" button will be enabled only after loading a DAT file.
5. If the "Update Firmware" button is clicked, the utility removes reference to a previously loaded DAT file and disables the "Update Device" button.
6. Any changes made to the YY-MM-DD-S-Sno controls will be lost if user switches to a different tab and returns to the "Production" tab. The changes should be saved immediately to a DAT file before switching tabs. If the user needs to switch tabs to make changes in those tabs, then those changes should be made first and lastly switch to the "Production" tab. Now the user can make necessary changes to the formatted serial number controls and can be saved to a DAT file that will include all the changes done on the other tabs as well. However, NOTE THAT THIS IS THE BEHAVIOR ONLY WHEN A DAT FILE HAS NOT BEEN LOADED ALREADY AND THE BUTTON "Update Device" IS DISABLED.
7. When a DAT file is loaded and the "Update Device" button is enabled, following behavior is to be expected;
 - a. Changes made to serial number controls in "Production" tab will be lost if user changes tabs. However, these changes will be active for the user to save to a DAT file or update to the device immediately after the changes are made and before switching tabs.
 - b. Changes made to controls on other tabs will be lost and can never be saved to a DAT file or updated to a device.
 - c. Changes made to serial number controls, though available for an immediate update to a device, will be lost after the update completes if those changes don't reflect the current date (YY-MM-DD).

Button Behaviour:

1. Update Device
 - a. Enabled only after a DAT file is loaded
 - b. Disabled whenever a firmware update is done
 - c. When clicked, app reads the serial number from the YY-MM-DD-S-Sno controls and embeds this serial number into the DAT file that is loaded in memory to write to the device.
 - d. After every update, the DAT file is updated to reflect the last serial number used to write to the device and also automatically increments the serial number.
2. Update Firmware
 - a. Prompts for a DFU file (only the first time) and uses this file to update the device's firmware.
 - b. Always ignores any changes done to the controls in all tabs. The utility always reads the descriptor data from the device and embeds this into the DFU file image before writing the DFU file image to the device. Thus, after a firmware update, the device's descriptor data should be the same as it was before the update.
 - c. If a DAT file was previously loaded, clicking this button would unload the DAT file and disable the "Update Device" button.
3. Load DAT File
 - a. Loads a DAT file into memory.
 - b. Enables the "Update Device" button
 - c. Any changes done to controls on all tabs are lost while switching tabs.
 - d. Changes done to YY-MM-DD-S-Sno controls are available for saving to a DAT file or writing to device only immediately after the changes are made.
 - e. When a DAT file is loaded, the YY-MM-DD-S-Sno controls are set to reflect current date, same station number digit as in the DAT file and either a default value of "00000" or DAT file's last 5 digits of serial number value incremented by one. The default value of "00000" is used whenever the DAT file's YY-MM-DD digits do not match the current date.
4. Save DAT File
 - a. Saves the values from the controls to a DAT file.
 - b. Refer to earlier sections to find out when changes to controls are lost.
 - c. After saving to DAT file, this DOES NOT automatically load that DAT file into memory.

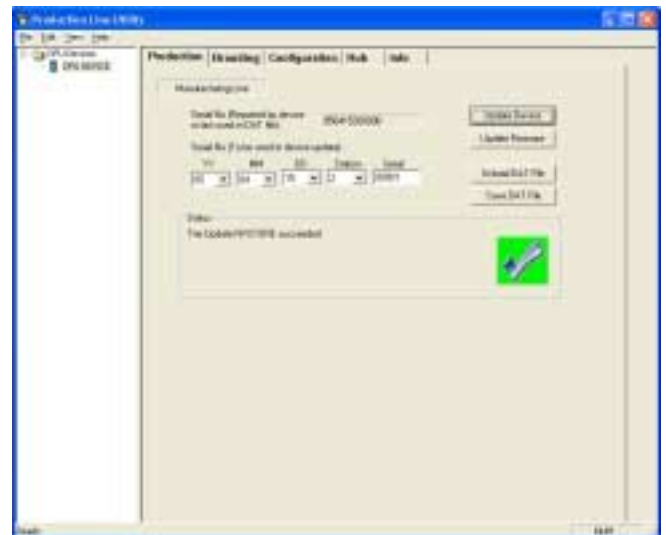
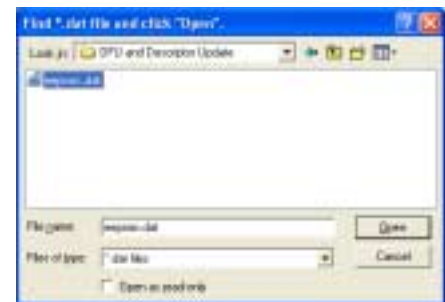
Setting Up the PLDU Application

1. First attach a USB2230 device to the host. To start the PLDU application, simply double click “PLDU.exe” executable.
2. After the main program dialog opens the production tab displays four options:
 - a. Update Device- Updates NVStore descriptor data such as VID/PID, Manufacturer and Product ID strings from the “EEPROM.DAT” file. Note – this option is not available until a DAT file is loaded.
 - b. Update Firmware- Updates the device firmware using a DFU update file with the .dfu extension.
 - c. Load DAT file – Loads a DAT file into memory
 - d. Save DAT file - Saves the values from the controls to a DAT file



Using the PLDU to Update Device Descriptors

1. The first operation that should be performed on a USB2230 device coming off the production line is to update its descriptors. To do this, first press “Load DAT file”. The application will prompt you to select the EEPROM.dat file that will be used to program the descriptors. Once the EEPROM.DAT file has been selected the option to Update Device will now be available.
2. Click the “Update Device” button. The PLDU application will swap the mass storage class driver for the SMSC DFU driver.
3. Once the DFU driver swap has completed, the data from the eeprom.dat file that is loaded is programmed into the device. The operation takes about 12 seconds to complete. Provided the programming was successful, the Status box displays a green box with a checkmark and reports success. At this point the user simply detaches the device and reattaches the next device to be programmed. The PLDU automatically updates the EEPROM.DAT file to the next unique serial number.



Using the Production Line Test Utility (PLTU)

Purpose: The PLTU application is used to test the basic functionality of USB2230 devices in a production line environment using Windows 2000 (SP3) only. The IRDA functionality of 2230 is currently only supported in Windows XP, however; this application can still be used in Windows 2000 to test the functionality of the Mass Storage functionality of the 2230. The application creates a subdirectory on the media for each LUN, copies a 'Test File' to the subdirectory, deletes the 'Test File', and then deletes the subdirectory.

Features:

1. Capable of testing 5 devices with 4 LUNs each simultaneously.
2. After testing, the application cleans up the registry entries involving the OEM's VID, PID, Inquiry MFG and Product strings.
3. Graphical and Text status display of test results.
4. GUI editor to edit and create ini files.

Creating the PLTU ini File

Before using the PLTU you must create or edit an ini file. A sample ini file is shipped with the PLTU application which can be modified for your setup. The ini file should contain the following lines:

OEMVID = ***VID***

This is the original equipment manufacturer's VID (Vendor ID) of the device whose descriptor has already been updated. The '***VID***' is specified as a four digit hexadecimal number.

OEMPID = ***PID***

This is the original equipment manufacturer's PID (Product ID) of the device whose descriptor has already been updated. The '***PID***' is specified as a four digit hexadecimal number.

INQUIRY_MFG = ***Inquiry MFG String***

This is the string returned by the device as part of the Vendor information in the Inquiry data. This can be of maximum 8 characters.

INQUIRY_PRODUCT = ***Inquiry Product String***

This is part of the string returned by the device Product information Inquiry data. This can be of maximum 5 characters.

TEST_FILE = ***path to Test file***

Specifies the full path to the file that is to be used during file copy tests.

DEV1_LUN0 = ***Drive Letter***

DEV1_LUN1 = ***Drive Letter***

DEV1_LUN2 = ***Drive Letter***

DEV1_LUN3 = ***Drive Letter***

DEV2_LUN0 = ***Drive Letter***

DEV2_LUN1 = ***Drive Letter***

DEV2_LUN2 = ***Drive Letter***

DEV2_LUN3 = ***Drive Letter***

DEV3_LUN0 = ***Drive Letter***

DEV3_LUN1 = ***Drive Letter***

DEV3_LUN2 = ***Drive Letter***

DEV3_LUN3 = ***Drive Letter***

DEV4_LUN0 = ***Drive Letter***

DEV4_LUN1 = ***Drive Letter***

DEV4_LUN2 = ***Drive Letter***

DEV4_LUN3 = ***Drive Letter***

Creating the PLTU ini File (Cont.)

DEV5_LUN0 = *Drive Letter*
DEV5_LUN1 = *Drive Letter*
DEV5_LUN2 = *Drive Letter*
DEV5_LUN3 = *Drive Letter*

These lines specify the Drives that are associated with the multiple LUNs of the respective devices to be tested. If the 'Drive Letter' is not specified for a particular LUN, then it means that the corresponding LUN of that device is NOT to be tested. If the 'Drive Letter' is not specified for all LUNs for a particular device, then it means that the entire device is either NOT present or NOT to be tested.

A Sample PLTU ini File

```
OEMVID      = 0424
OEMPID      = 2230
INQUIRY_MFG  = SMSC
INQUIRY_PRODUCT = 2230
TEST_FILE    = C:\TEST\1MEG.R01
```

```
DEV1_LUN0 = F
DEV1_LUN1 = G
DEV1_LUN2 = H
DEV1_LUN3 = I
```

```
DEV2_LUN0 = J
DEV2_LUN1 = K
DEV2_LUN2 = L
DEV2_LUN3 = M
```

```
DEV3_LUN0 = N
DEV3_LUN1 = O
DEV3_LUN2 = P
DEV3_LUN3 = Q
```

```
DEV4_LUN0 = R
DEV4_LUN1 = S
DEV4_LUN2 = T
DEV4_LUN3 = U
```

```
DEV5_LUN0 =
DEV5_LUN1 =
DEV5_LUN2 =
DEV5_LUN3 =
```

NOTE:

There can be spaces before and after the '=' sign, but the total number of characters for an entire line (including spaces) should be less than 255.

Setting Up the PLTU Application

1. First attach a USB2230 device to the host. To start the PLTU application, simply double click “TestDevice.exe” executable. The application will prompt you to select the location of the ini file when it is first started.

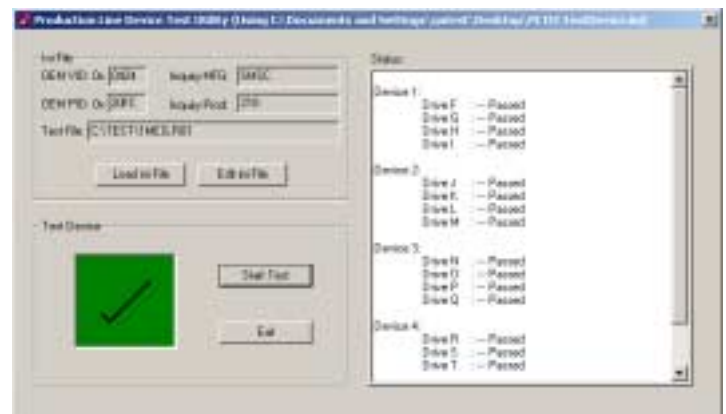


2. Provided the ini file contains the correct path to the key files on the local machine, the main program dialog opens. The station is now ready to begin testing devices. At this point you should attach the devices to be tested and ensure that they have good media with sufficient free space to hold the file being used for testing.



Using the PLTU to Test Multiple Devices

1. Once all of the devices have been attached, the user simply presses the “Start Test” button to begin testing devices in accordance with the contents of the ini file being used. After the testing has completed, the user receives a graphical representation of the test results in the form of a green box with a black checkmark to indicate “PASS”, or a red box with a black “X” to indicate “FAIL”.



2. Once the test has completed, the user should remove all of the tested devices and then attach the next set of devices to be tested. Once all of the devices are attached and enumerated (as indicated by the presence of drive icons in Windows Explorer), the user repeats step 1 to test the next set of devices.

Known Issues with the USB2230 Production Line Utilities

Issue:	Workaround:	Status:
The PLDU and PLTU applications are designed to be used with Windows 2000 (SP3) host systems using the Microsoft mass storage class driver. While the applications may work with other operating systems, only Windows 2000 (SP3) is supported.	N/A	N/A
Some EHCI host controller drivers such as Orange Micro's do not work properly with the DFU driver swapping performed by the PLDU and PLTU applications.	We highly recommend that you use the Microsoft supplied EHCI drivers for the test systems running the PLDU and PLTU applications.	N/A
The PLTU does not distinguish between general device write failures and media specific write failures. This means that the test will fail if no media present in the drive, media is full, media is unformatted, media is corrupt, media is write protected, etc.. Under such circumstances, the test results do not reflect the status of the device, but rather the failure of the media. Hence, it is recommended that the test is performed again on the device with known good media.	Only use known good media to perform the PLTU testing.	N/A
Due to caching by the OS, the IO transfer may not be fully completed before the test results are displayed by the application. It is recommended that the user wait for 5 to 10 seconds before disconnecting the devices.	Wait 5-10 seconds after completion of the PLTU tests before removing the devices from the host.	N/A
User may experience errors when running applications if certain files used by applications are marked read only.	Make sure that all files used with these utilities are not marked read only	N/A
PLDU displays a program error when attempting to update the firmware using an external version of the 2230 firmware.	Use USBDM application for device firmware updates. PLDU may still be used for Descriptor Updates.	Will be fixed in future release.

Using the QuickTest Production Line Read/Write Test Utility

The QuickTest utility is a streamlined version of the full Production Line Test Utility discussed previously. QuickTest can test a maximum of (4) USB2230 devices at a time, with a maximum of 4 LUNs each. The testing procedure is very simple involving these only 4 steps:

1. Writes to media on each LUN starting from LBA 1024
2. Reads from media on each LUN starting from LBA 1024
3. Compares the data read against the data written to the media
4. Updates the status for each LUN in the application



The testing is performed on all the LUNs of the device serially. However, tests on multiple devices are performed simultaneously using multiple threads. The QuickTest utility requires the presence of the SMSC password filter driver to send BULK-ONLY commands, totally by-passing the native file system. On windows 2000 systems, Service Pack 3 should be installed. The IRDA functionality of 2230 is currently only supported in Windows XP, however; this application can still be used in Windows 2000 to test the functionality of the Mass Storage functionality of the 2230.

QuickTest.exe requires the SMSC password filter driver (Smscpwd.inf) to be installed in order to function properly. This driver is no longer installed by the card reader installer and must be installed manually before running QuickTest. In order to install the password filter drivers, copy Smscpwd.inf and Smscpwd.sys to your system. Open the device manager and double click on the USB Mass Storage Device entry in the Universal Serial Bus Controllers section. In the driver tab, select update driver. The wizard will assist in installing this driver. When given the choice, specify to have the wizard display a list of known drivers for this device. Choose "Have Disk" and browse to where you copied the smscpwd.inf file and select it. This should give you the option to install "USB Mass Storage Device with Password Protection (WinMe)".

Limitations of the QuickTest Utility:

1. Does not distinguish between general device write failures and media specific write failures. This means that the test will fail if no media is present in the drive, the media is full, unformatted, corrupt, write protected, etc.. Under such circumstances, the test results do not reflect the status of the device. Hence, it is recommended that the test is performed again on the device with known good media.
2. The time taken to complete the tests depend on the following:
 - Test size - This can be from 64KB to 5000KB. The bigger the size, the more time it will take to complete the tests.
 - Number of devices connected- The field "Max Devices" specifies how many devices to test at once (should be $1 \leq N \leq 4$). However, it is not necessary that the actual number of devices connected be equal to the number specified in the "Max Devices" field. For example, the "Max Devices" field can specify 4 but the actual number of devices connected may be ≤ 4 or > 4 . However, the utility will either test only the actual number of devices connected or the "Max Devices", whichever is less. Though tests on multiple devices are performed simultaneously, the time taken for the tests to complete on multiple devices will be a little more than that for a single device.

Using the EPRMUPDT.exe Utility

EPRMUPDT.exe is a DOS based utility used to write and / or read EEPROM data to / from the USB2230 device. This utility is designed to be used by OEMs in a production line environment with as little human intervention as possible.

EprMUpdt Usage:

```
EprMUpdt [-h|-u] [-v] [-c] [-w"oFileName"] [-r"iFileName"] [-t"HostController"]
          [-l"LogFileName"] [-p"xxxxyyyy"]
-h|-u .....print help/usage
-v .....verbose, optional, default is off
-c .....confirm scanned serial number (last 3 digits) before updating EEPROM
-w"oFileName".....name of DAT file (with full path) that is to be written to device EEPROM
-r"iFileName".....name of formatted text file (with full path) that is to be created by reading device EEPROM
-l"LogFileName".....log the serial number to the specified log file
-t"HostController"...specifies the host controller type to which the device is attached. This should be "UHCIn",
                      "OHCIn" or "EHCIIn", where 'n' is a number (0 to 9) specifying the host controller in the enumeration order.
                      This is an optional parameter and if not specified, a default value of "UHCI" will be used. Similarly 'n' is also
                      optional and if it is not specified, a default value of '0' will be used.
-p"xxxxyyyy".....specifies that multiple MSC devices may be connected to the system and that the utility should
                      find the device by matching the specified Vid and Pid. The value "xxxx" denotes the Vid and the value
                      "yyyy" denotes the Pid. Both the Vid and Pid are to be specified as 4 digit HEX numbers. The -p option is
                      optional and if not specified, or contains an invalid value, then the utility would default to finding the first
                      MSC device using the class, subclass and protocol values.
-i .....infinite loop, till user presses 'CTRL C' to quit
```

Note:

1. All options can be specified using both UPPERCASE or lowercase letters.
2. The double quotes (") around file names for -w, -r and -l options is optional. If the path names does not contain blank spaces, then the double quotes are not necessary. If the path names contain blank spaces, then the double quotes are mandatory.
3. The file names for the -w, -r and -l options are to be specified with full path information. If the files are in the current directory, then the path information is not necessary.
4. The double quotes around the 'HostController' in -t option is optional.

Features:

1. Uses a template EEPROM.DAT file, modifying the serial number alone by scanning it off the keyboard buffer, to update the device EEPROM.
2. Reads the contents of the device EEPROM and generates a formatted text file that vividly describes all the fields of EEPROM structure.
3. The options for writing and reading EEPROM data can be specified together or alone.
4. Provides an option (-c) to confirm the scanned serial number (last 3 digits) with the user before updating the EEPROM data.
5. Provides an option (-v) to turn on or off the additional debug / status comments.
6. Provides an option (-l"LogFileName") to log the serial number to the user specified log file.
7. Allows processing devices one after another in a loop till user wants to exit (by pressing 'Ctrl C') by specifying the -i option in the command line. Otherwise, the utility will exit back to the command prompt after it is done with a single device.

Using the EPRMUPDT.exe Utility (cont.)

8. Displays the status by showing a big "ERR", "FAIL" or "PASS" along with other relevant information.

"ERR" - Means an error occurred outside of the main process of updating or reading to / from the device. This can happen if there are any errors while parsing the input arguments, or invalid usage, or invalid file paths, or any errors while starting the host controller and root hub. The application will exit with code 2 during such circumstances.

"FAIL" - Means an error occurred during the process of updating or reading to / from the device. This can happen if no matching devices are found, or verification of last 3 digits of serial number fails, or error while writing data to device, or error while reading data from device, or verification of read and write data fails. The actual reason for the failure is given below the "FAIL" status and the application exits with code 1 during such circumstances. If the -i option is specified, then the application proceeds to prompt for scanning the serial number again. At this point, it is left to the user discretion, whether to connect a new device or proceed with the existing device. For example, if the failure is due to last 3 digits serial number mismatch, it could be due to human error rather than a device error and so the user may want to proceed with the same device again.

"PASS" - Means no error occurred and the process of updating or reading to / from the device completed successfully, including all necessary verifications and the application exits with code 0. If the -i option is specified, then the application proceeds to prompt for scanning the serial number again. At this stage, the user can safely remove the existing device and connect a new device and enter the serial number again.

9. The utility will return with one of the following exit codes.

- 0 - Indicates "PASS"
- 1 - Indicates "FAIL"
- 2 - Indicates "ERR"

10. The utility will work with all types of host controllers (UHCI, OHCI & EHCI) and the host controller to which the device is connected is specified by the -t option. The -t option specifies the type of the host controller as well as the number in the PCI enumeration order of the host controllers. These two together identify a unique host controller which the application enumerates to detect the test device. Note that this is optional and that the default values will be used if it is not specified.

examples:

- | | |
|-----------|--|
| -t"UHCI" | - Test on the 1st UHCI host controller |
| -t"EHCI0" | - Test on the 1st EHCI host controller |
| -t"OHCI2" | - Test on the 3rd OHCI host controller |

11. The utility was searching for the MSC device by looking in to the DeviceClass, DeviceSubClass and DeviceProtocol values of all connected devices and using the first matched device for further operations. This led to limitations like the "The MSC device that is to be processed by the utility has to be the first MSC device in enumeration order". Moreover, the Class, SubClass and Protocol values of a MSC device are 0, 0 and 0 which are the values of any USB composite class device. This meant that the limitation had to be extended further as "The MSC device that is to be processed by the utility has to be the first Composite class device in enumeration order". This was modified in v2.0 of the utility so that the utility can now find a device by searching for its VendorId (Vid) and ProductId (Pid). This is accomplished by using the "-p" option and specifying the Vid and Pid as 4 digit Hex numbers. For backwards compatibility reasons, this is provided as an optional option. Test systems which do not have any other composite class or MSC devices need not specify this option.

Limitations of the EPRMUPDT.exe Utility:

1. Supports devices connected only at the root hub level.
2. When "-p" option is specified with a Vid and Pid and multiple devices of same Vid and Pid are connected to the system, the utility will process only the first device in the enumeration order.
3. There is no bus traffic after SPT EEPROM write call. Still, the EEPROM write call should pass and the application will report the status of the write. Use only one of the -w or -r options and don't combine both the options. If both the options are used, then the write should pass but the read should fail because of a known issue.

Using the Windows XP Special Memory Stick Format Registry Key

Windows XP has the capability to apply a Sony factory format on Memory Stick cards by adding a special key to the registry:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\PerHwIdStorage\
USBSTOR#DiskSMSC____2230_U_HS-MS____] "DeviceGroup"="MemoryStick"
```

This key has to be customized to match the inquiry data returned from the device. The inquiry data is made up of the first 8 characters of the Manufacturer String, followed by the first 5 characters on the Product String. In the example registry key above, the strings are:

Manufacturer String = "SMSC" (Note that SMSC is followed by four spaces denoted by underscores to make up the 8 characters.)

Product String = "2230 USB2230" (Note that only the first 5 characters, including the space, are used.)

This registry key works for Windows XP only. It will not work for any other operating system. Once the registry key has been added, when a user formats a Memory Stick card from using Windows, the Sony factory FAT format will be applied, including the creation of the "MEMSTICK.IND" hidden file.

Using the KillReg Utility

KillReg is a DOS based application to stop a device and clean its related registry entries during an automated uninstallation process. KillReg is designed to be called from a Windows Installer script. It is used during installation and uninstallation of USB97C210/223/2224/2228/2230 devices under all Windows operating systems to remove the device entries from the registry. This allows the SMSC Win2K or Windows native driver to be loaded if the device has previously been installed without a driver, or with an incorrect driver. KillReg is also used during the uninstallation process to completely remove the registry entries for a particular device.

Requirements:

KillReg requires an ini file to be present in the Windows directory. The name of this ini file should be passed as command line argument to the application from the installer script.

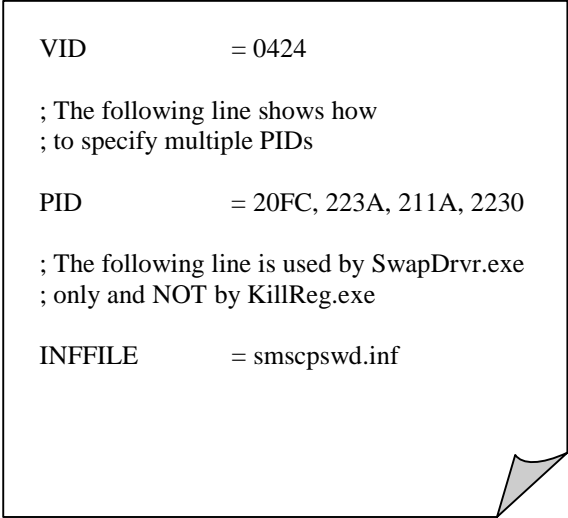
INI File Requirements:

1. The ini file should be in the Windows directory.
2. The ini file should contain the following lines;

```
VID = VID  
PID = PID1[,PID2,PID3,...,PID30]
```

where VID and PID are represented as 4 digit hexadecimal numbers.

A Sample ini File:



```
VID          = 0424  
  
; The following line shows how  
; to specify multiple PIDs  
  
PID          = 20FC, 223A, 211A, 2230  
  
; The following line is used by SwapDrvr.exe  
; only and NOT by KillReg.exe  
  
INFFILE      = smscpswd.inf
```

NOTE:

1. The ini file is also used by the application "SwapDrvr.exe", which will expect the line specifying the INFFILE. KillReg ignores this line.
2. Multiple PIDs separated by a comma can be specified to uninstall all the PIDs associated with a single VID.

Using the Swapdrv Utility

Swapdrv is a DOS based application used by a Windows installer to load the password filter driver in Windows XP. The only USB2230 application that requires the password filter driver be loaded when running XP is the QuickTest production line test utility. The Cardreader Software Installer Safe Removal.exe also requires the use of Swapdrv. If you are not using these utilities or do not want to include it in your installer, you can skip this section.

Requirements:

1. The device should be connected while this application is invoked from a Windows installer. The application will prompt the user to connect the device during run time.
2. Swapdrv needs an ini file to be present in the Windows directory. The name of this ini file should be passed as command line argument to the application from the installer script.
3. The installer application should have already placed the required INF and SYS files in their correct locations.

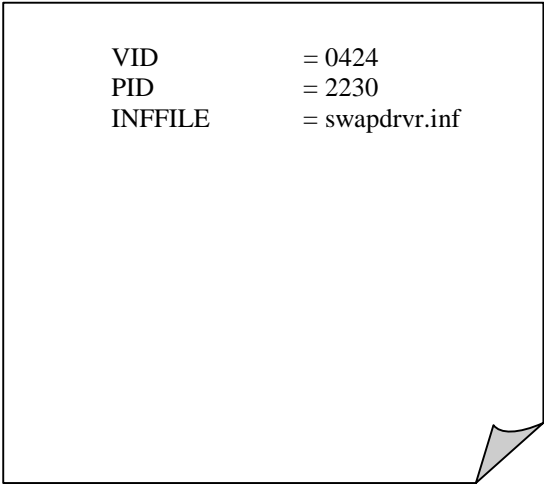
INI File Requirements:

1. The ini file should be in the Windows directory.
2. The ini file should contain the following lines;

```
VID = VID  
PID = PID  
INFFILE = Inf file name
```

where VID and PID are represented as 4 digit hexadecimal numbers.

A Sample ini File:



```
VID          = 0424  
PID          = 2230  
INFFILE      = swapdrv.inf
```

Using the Dos Production Line Utility (DosPLTU)

DosPLTU is a DOS based utility intended to be used by OEMs to streamline their production testing, requiring as little human intervention as possible. This utility supports checking the device firmware version, checking and / or updating the device EEPROM with a template DAT file, and performing R/W tests on all the logical units (LUNs) supported by the device.

DosPLTU Usage:

```

DosPLTU [-h|-u] [-v] [-f"version"] [-t -n"loopcount" -s"testsize"]
        [-e"DATFileName" | -w"DATFileName" | -x"DATFileName"]
        [-l"LogFileName"] [-c"HostController"] [-d"CfgFileName"]
        [-p"xxxxyyyy"]
-h|-u .....print help/usage
-v .....verbose, optional, default is off
-f"version".....version number that is to be checked against the firmware version of the device
-t .....perform R/W tests
-n"loopcount".....specifies the number of times the R/W tests are to be performed. This is optional and a default
        value of 10 will be used if this is not specified
-s"testsize".....specifies the test transfer size in KB for the R/W tests. This is optional and a default value of 64KB
        will be used if this is not specified
-d"CfgFileName".....name of the configuration file (with full path) that specifies the media types for each supported
        LUN. This is optional and if not specified, then testing would be done on all LUNs for one media type, with
        out prompting the user to insert other types of media
-e"DATFileName".....name of DAT file (with full path) that is to be checked against the device EEPROM. This
        option cannot be specified with -w or -x options
-w"DATFileName".....name of DAT file (with full path) that is to be written to device EEPROM This option cannot
        be specified with -e or -x options
-x"DATFileName".....name of DAT file (with full path) that is to be checked against the device EEPROM and if
        necessary that is to be written to the device EEPROM. This option cannot be specified with the -e or -w
        options
-l"LogFileName".....name of the log file (with full path) to which the test status messages are logged
-c"HostController"....specifies the host controller to which the device is attached. This should be "UHCIn", "OHCIn"
        or "EHCIn", where 'n' is a number (0 to 9) specifying the host controller in the enumeration order. This is an
        optional parameter and if not specified, a default value of "UHCI" will be used. Similarly 'n' is also optional
        and if it is not specified, a default value of '0' will be used.
-p"xxxxyyyy".....specifies that multiple MSC devices may be connected to the system and that the utility should
        find the device by matching the specified Vid and Pid. The value "xxxx" denotes the Vid and the value
        "yyyy" denotes the Pid. Both the Vid and Pid are to be specified as 4 digit HEX numbers. The -p option is
        optional and if not specified, or contains an invalid value, then the utility would default to finding the first
        MSC device using the class, subclass and protocol values.
-i .....infinite loop, till user wants to quit

```

Note:

1. All options can be specified using both UPPERCASE or lowercase letters.
2. The double quotes (") around file names is optional. If the path names does not contain blank spaces, then the double quotes are not necessary. If the path names contain blank spaces, then the double quotes are mandatory.
3. The file names are to be specified with full path information. If the files are in the current directory, then the path information is not necessary.
4. The double quotes around the 'version' in -f option is optional.
5. The value of 'version' is specified as a max 4-digit decimal integer number.
6. The double quotes around the 'HostController' in -t option is optional.
7. The double quotes for -n and -s options are optional.

Option Groups and Priority Levels:

1. The options are classified into 5 groups as described below.

a. Usage	- "-h" or "-u"
b. Firmware check	- "-f"
c. EEPROM check / update	- "-e", "-w" and "-x"
d. R/W tests	- "-t", ["-n", "-s" and "-d"]
e. Miscellaneous	- "-v", "-c", "-l", "-p" and "-i"
2. The utility has a priority level for each group of options. The priority level and processing details are described below.
 - a. Usage group - Has the highest priority (level 0). If this is specified, then the utility would just display the program usage and exit. All other options are ignored and are not processed.
 - b. Firmware check group - Has the next highest priority (level 1). The utility processes this option before EEPROM check and R/W test options. If the device firmware does not match the version specified with this option, then the utility would display an error message and exit without processing any other option.
 - c. EEPROM check / update group - Has a priority level of 2. If "-f" option is specified, the utility would process this option after successfully checking the device firmware version. Otherwise, this would be processed first. It is important to note that this group has 3 options ("-e", "-w" and "-x") which are mutually exclusive. That is, only one of the 3 options can be specified. If any error occurs while processing this group, the utility ignores the R/W test option and exits after displaying the corresponding error message.
 - d. R/W test group - This has the lowest priority (level 3) and is processed last after successfully processing other specified options. This group has three additional options ("-n", "-s" and "-d") that may or may not be specified. Refer to the usage for more details about these options.
 - e. Miscellaneous group - Has no priority level at all. The options under this group are very general and only help the user to control how the tests are done and how the results are displayed. These options, by themselves, do not affect the types of tests or their order in any way.

DosPLTU Features:

1. Checks the firmware version of the device.
2. Checks the device EEPROM against a template DAT file and returns an error if any mismatch is found. This is achieved by using the "-e" option and is useful in testing devices whose EEPROM has already been updated.
3. Updates the device EEPROM always with a template DAT file with out checking for any mismatch. After every update, the serial number is automatically incremented and the DAT file is updated. This is achieved by using the "-w" option and is useful in updating the device EEPROM for the first time.
4. Checks the device EEPROM against a template DAT file and updates the device EEPROM if any mismatch is found. If the EEPROM is updated, the serial number is automatically incremented and the DAT file is updated. This is achieved by using the "-x" option and is useful in testing devices whose EEPROM may or may not have been already updated.
5. Performs R/W tests on all LUNs supported by the device. The tests are performed using the loop count and test size values specified with "-n" and "-s" options. The R/W test option also takes in an optional -d option that specifies the device configuration. When this option is specified, the tests are performed on each LUN prompting the user to insert the supported media types for the LUN. This option is useful in cases, where LUN sharing is done in devices by having a combo socket and it is necessary to test all the media types supported by the socket. For a sample device configuration file, please look in to "Device.cfg" file.
6. Provides an option (-v) to turn on or off the additional debug / status comments.
7. Provides an option (-l"LogFileName") to log all messages to the user specified log file.
8. Allows processing devices one after another in a loop till user wants to exit by specifying the "-i" option in the command line. Otherwise, the utility will exit after it is done with a single device.
9. Displays the status by showing a big "ERR", "FAIL" or "PASS" along with other relevant information.

"ERR" - Means an error occurred outside of the test process. This can happen if there are any errors while parsing the input arguments, or invalid usage, or invalid file paths, or any errors while starting the host controller and root hub.

The application will exit with error code 1 during such circumstances.

"FAIL" - Means an error occurred during the process of testing. This can happen if no matching devices are found or any of the test fails. The actual reason for the failure is given below the "FAIL" status.
The application will exit with error code > 1 during such circumstances.

"PASS" - Means no error occurred and the process of testing completed successfully.
The application will exit with error code 0 during such circumstances.

10. The utility will work with all types of host controllers (UHCI, OHCI & EHCI) and the host controller to which the device is connected is specified by the -c option. The -c option specifies the type of the host controller as well as the number in the PCI enumeration order of the host controllers. These two together identify an unique host controller which the application enumerates to detect the test device. Note that this is optional and that the default values will be used if it is not specified.

examples:

-c"UHCI"	- Test on the 1st UHCI host controller
-c"EHCI0"	- Test on the 1st EHCI host controller
-c"OHCI2"	- Test on the 3rd OHCI host controller

Note: In order to properly specify the number in the PCI enumeration order of the host controllers the end user has to know how many host controllers of the given type are present in the system and also the enumeration order of the host controller to which the device is attached. If these details are not known, this information can be found by trial and error methods.

11. The utility will return with one of the following exit codes.

- 0 - Indicates "PASS"
- 1 - Indicates "ERR"
- 2 - Indicates "FAIL" (Device not found error)
- 3 - Indicates "FAIL" (Firmware mismatch error)
- 4 - Indicates "FAIL" (Error while reading device EEPROM)
- 5 - Indicates "FAIL" (Device EEPROM and template DAT file mismatch error)
- 6 - Indicates "FAIL" (Error while writing to the device EEPROM)
- 7 - Indicates "FAIL" (Error verifying updated EEPROM data)
- 8 - Indicates "FAIL" (Error while initializing disk(s) for R/W tests)
- 9 - Indicates "FAIL" (Error while writing to disk)
- 10 - Indicates "FAIL" (Error while reading from disk)
- 11 - Indicates "FAIL" (Error verifying read and write data)
- 12 - Indicates "FAIL" (Error creating the log file)

12. The utility was searching for the MSC device by looking in to the DeviceClass, DeviceSubClass and DeviceProtocol values of all connected devices and using the first matched device for further operations. This lead to limitations like the "The MSC device that is to be processed by the utility has to be the first MSC device in enumeration order". Moreover, the Class, SubClass and Protocol values of a MSC device are 0, 0 and 0 which are the values of any USB composite class device. This meant that the limitation had to be extended further as "The MSC device that is to be processed by the utility has to be the first Composite class device in enumeration order". This was modified in v2.0 of the utility so that the utility can now find a device by searching for it's VendorId (Vid) and ProductId (Pid). This is accomplished by using the "-p" option and specifying the Vid and Pid as 4 digit Hex numbers. For backwards compatibility reasons, this is provided as an optional option. Test systems which do not have any other composite class or MSC devices need not specify this option.

NOTE:

As mentioned above, when the device EEPROM is updated, the DAT file is updated with the serial number incremented by one. During such cases, there is a chance for the serial number to overflow from "FFFFFFFFFFFF" to "000000000000". When this overflow occurs, there will be a warning displayed to indicate the overflow. However, the testing on the current device continues normally as the overflow will matter only with the next device that is to be tested. Even if the tests on the current device pass successfully, the return value will be "ERR" to indicate the serial number overflow error.

Limitations of the DosPLTU.exe Utility:

1. Supports devices connected only at the root hub level.
2. In order to properly specify the number in the PCI enumeration order of the host controllers the end user has to know how many host controllers of the given type are present in the system and also the enumeration order of the host controller to which the device is attached. If these details are not known, this information can be found by trial and error methods.
3. The utility does not distinguish between actual device failures and media specific failures during R/W tests. Hence, it is recommended that the R/W tests are done on devices with known good media.
4. It is recommended that no other USB devices are connected to the system, specially when the system is booting.
5. It is recommended that the utility is used on systems having Pentium II or III processors (400 - 800 MHz processor speed). The utility seems to fail more as the processor speed increases. On systems having Pentium4 processors with speed as high as 1.4 to 2.4 GHz, the utility works reliably around 80% of the times and varies with different system configurations.

Device Configuration File Structure:

 The device configuration file is used with the "-d" option for performing R/W tests on all supported media types of each LUN of the device. The utility prompts the user to insert different media, one by one, in order to perform the tests. For this, the utility needs to know the device configuration, ie. how many LUNs the device supports and the different media types supported by each LUN. These are specified in the device configuration file.

The number of LUNs supported by the device is indicated by the following line;

MAX_LUNS=n
 where 'n' is a number (> 0 and <= 4) that specifies the number of LUNs

The media types supported by each LUN are indicated as shown below;

Lx=t1,t2,...,t6
 where 'x' is a zero based LUN number, that should be < 'n' specified above
 and 't1','t2',..., 't6' are media types that should be one of the valid media types.

The valid media types that are defined, for now (more could be added later), are given below;

1 = CF 2 = MS 3 = SM 4 = XD 5 = SD 6 = MMC

Note:

-
1. There should be no spaces before and after the equals ('=') sign
 2. Number of LUNs specified is 1 based, ie. if the device supports 4 LUNs, specify MAX_LUNS=4
 3. The LUN index ('x') is 0 based, ie. 1st LUN is indexex as L0.
 4. Multiple media types are separated by commas(',') without any spaces in between as shown below
 L0=1
 L1=2
 L2=3,4
 L3=5,6

The utility parses this file to understand the device configuration. From the example file shown above, the utility understands that the device supports 4 LUNs and the 1st LUN supports only CF media, the 2nd LUN supports only MS media, the 3rd LUN supports SM and XD media and the 4th LUN supports SD and MMC media.

Media Tested with the USB2230

The following flash media cards were used during the development and testing of the USB2230. All media listed has been determined to work properly and be compatible with the USB2230.

Compact Flash	Memory Stick	Secure Digital	Mobile MMC
CompUSA 16MB	Lexar 16MB	IO Data 64MB	Samsung 128MB
CompUSA 48MB	Lexar 32MB	Buffalo 256MB	
CompUSA 64MB	Lexar 64MB	Kingston 256MB	Smart Media
Delkin Devices Pro 640MB	Lexar 128MB	Lexar 16MB	Fuji Film 8MB
Hyperstone 8MB	Lexar 256MB	Lexar 32MB	Fuji Film 16MB
IO Data 4MB	PQI 64MB	Lexar 64MB	Kingston 64MB
IO Data 8MB	PQI 128MB	Lexar 128MB	I-O Data 8MB
IO Data 32MB	SanDisk 16MB	Lexar 256MB	I-O Data 16MB
King Max 8MB	SanDisk 64MB	Lexar 512MB	I-O Data 32MB
King Stone 64MB	SanDisk 128MB	Memorex 32MB	I-O Data 64MB
Lexar 16MB	Sony 8MB	Memorex 128MB	I-O Data 128MB
Lexar 32MB	Sony 16MB	Panasonic 512MB	Lexar 16MB
Lexar 48MB	Sony 32MB	PNY 64MB	Lexar 32MB
Lexar 64MB	Sony 64MB	PNY 128MB	Lexar 64MB
Lexar 128MB	Sony 128MB	PNY 512MB	Lexar 128MB
Lexar 256MB	Sony 256MB	PQI 64MB	Memorex 32MB
Lexar 512MB (24x)		PQI 128MB	Memorex 64MB
Lexar 512MB (80x)	High Speed Memory Stick	PQI 256MB	Memorex 128MB
Lexar 1GB (4x)	Sony 16MB	SanDisk 32MB	Olympus 8MB
Lexar 1GB (24x)	Sony 32MB	SanDisk 64MB	PNY 128MB
Lexar 2GB (40x)	Sony 128MB	SanDisk 128MB	Samsung 32MB
Memorex 32MB		SanDisk Extreme 256MB	SanDisk 32MB
Memorex 64MB	Memory Stick Pro	SanDisk 512MB	SanDisk 64MB
Memorex 128MB	Sony 256MB	SanDisk 2GB	SanDisk 128MB
Memorex 256MB	Sony 512MB	SanDisk Ultra II 2GB	Viking 64MB
PNY 64MB	Sony 1GB	SimpleTech 128MB	
PNY 128MB	SanDisk 256MB		
PNY 256MB	SanDisk 512MB	High Speed Secure Digital	xD Picture Card
PNY 512MB	SanDisk 1GB	Panasonic 512MB	Fuji 32MB
PQI 16MB		Panasonic 1GB	Fuji 64MB
PQI 64MB	Memory Stick Duo		Fuji 128MB
Samsung 128MB	Sony 16MB	MMC	Fuji 256MB
SanDisk 16MB	Sony 32MB	Lexar 16MB	Fuji 512MB
SanDisk 32MB	Sony 64MB	Lexar 32MB	Olympus 16MB
SanDisk 1GB	Sony 128MB	Lexar 64MB	Olympus 32MB
SanDisk Extreme 1GB		PQI 32MB	Olympus 64MB
SanDisk Ultra 128MB	Mini Secure Digital	PQI 64MB	Olympus 128MB
Viking 32MB	Panasonic 32MB	PQI 128MB	Olympus 256MB
	Panasonic 64MB	PQI 256MB	Olympus 512MB
	Panasonic 128MB	SanDisk 8MB	PQI 64MB
IBM MicroDrive	SanDisk 32MB	SanDisk 16MB	
IBM Microdrive 340MB	SanDisk 64MB	SanDisk 32MB	
IBM Microdrive 1GB	Toshiba 32MB	SanDisk 64MB	
Hitachi 2GB		SimpleTech 64MB	
Magicstor 2.2GB		SimpleTech 128MB	

USB2230 Performance Benchmarks

The measurements were performed using HDBench v3.30 on a Windows XP (SP1) system with an ICH4 south bridge. (Pentium 4, 1.8GHz, 512MB DDR). All benchmarks were measured on new (out of the box) media. Please note that the benchmark performance of flash cards varies widely from manufacturer to manufacturer, and the performance of all manufacturers' cards degrade with use. In order to duplicate the results below, you must use brand new media and test on a similarly configured host.

USB2230 Benchmark Data

Full Speed (USB1.1)	Reads	Writes
<i>Compact Flash</i>	1043 KB/s	932 KB/s
<i>Memory Stick</i>	909 KB/s	550 KB/s
<i>High Speed Memory Stick</i>	811 KB/s	652 KB/s
<i>Memory Stick Pro</i>	1031 KB/s	902 KB/s
<i>Smart Media</i>	977 KB/s	537 KB/s
<i>XD Card</i>	818 KB/s	437 KB/s
<i>Secure Digital</i>	1039 KB/s	945 KB/s
<i>High Speed Secure Digital</i>	913 KB/s	906 KB/s
<i>Multimedia Card</i>	996 KB/s	374 KB/s
<i>High Speed Multimedia Card</i>	882 KB/s	811 KB/s

Media Used for Testing:

SanDisk Ultra II 1GB
 Lexar Media 128MB
 Sony MagicGate 128MB
 Sony 512MB
 Memorex 128MB
 Fuji xD-Picture Card 512MB
 SanDisk Extreme 256MB
 Panasonic Pro High Speed 512MB
 Lexar Media 64MB
 Samsung MMC Mobile 128MB

High Speed (USB2.0)	Reads	Writes
<i>Compact Flash</i>	9682 KB/s	5953 KB/s
<i>Memory Stick</i>	1540 KB/s	833 KB/s
<i>High Speed Memory Stick</i>	4031 KB/s	897 KB/s
<i>Memory Stick Pro</i>	4027 KB/s	3157 KB/s
<i>Smart Media</i>	4762 KB/s	1746 KB/s
<i>XD Card</i>	3968 KB/s	669 KB/s
<i>Secure Digital</i>	7275 KB/s	5340 KB/s
<i>High Speed Secure Digital</i>	10400 KB/s	7700 KB/s
<i>Multimedia Card</i>	1522 KB/s	486 KB/s
<i>High Speed Multimedia Card</i>	5696 KB/s	4442 KB/s

Media Used for Testing:

SanDisk Ultra II 1GB
 Lexar Media 128MB
 Sony MagicGate 128MB
 Sony 512MB
 Memorex 128MB
 Fuji xD-Picture Card 512MB
 SanDisk Extreme 256MB
 Panasonic Pro High Speed 512MB
 Lexar Media 64MB
 Samsung MMC Mobile 128MB

GPIO Assignment Table

The following is a table of GPIO assignments for the USB2230. Please note that multi-function GPIO operations are determined by attribute settings. Please refer to the software release notes for detail on configuration settings.

Name	Description	Function
GPIO0	<i>Not available due to pin count</i>	
GPIO1	Flash Media Activity LED	Media Activity LED
GPIO2	EE_CS	EE_CS
GPIO3	V_BUS	V_BUS
GPIO4	EE_DIN/EE_DOUT	EE_DIN&DOUT/xD Card Identification
GPIO5	SD Card Detect	SD Card Detect
GPIO6	A16 (external ROM only) /ROMEN	ROMEN/A16
GPIO7	EE_CLK/Unconfigured LED	EE_CLK/Uncfg LED
GPIO8	MS Power Control	MS Power Control
GPIO9	CF Power Control	CF Power Control
GPIO10	SM Power Control	SM Power Control
GPIO11	SD Power Control	SD Power Control
GPIO12	MS Activity/Transceiver ShutDown	MS Activity/Transceiver ShutDown

Known Firmware Related Issues**General:**

Issue:	Workaround:	Status:
If version 436 firmware is run externally (not from internal ROM) SIR speeds fail to meet acceptable benchmarks	None.	This firmware release is only for internal ROM. This will be fixed in the next external firmware release.

CF Devices:

Issue:	Workaround:	Status:
No known issues.		

MS Devices:

Issue:	Workaround:	Status:
When High Speed Magic Gate Memory Stick media is formatted with a FAT file system on a MacOS 10.X host, the media becomes unreadable on machines with Windows operating systems, but will continue to work normally with Macs.	None.	We believe this is a Magic Gate security protocol issue. We will continue to investigate and provide a fix in a future release of the USB2230 firmware if possible.

SM Devices:

Issue:	Workaround:	Status:
Writes to 2MB Smart Media cards are not supported.	None.	2MB Smart Media cards can be read by the USB2230, but writes are not supported. These cards are considered obsolete and there are no plans to implement support for them in the future.

SD/MMC Devices:

Issue:	Workaround:	Status:
Under certain conditions, the USB2230 device may fail to recognize an SD/MMC card inserted while writing to either CF or MS or SM cards.	Attempt to reinsert the card.	Currently under investigation. May be fixed in a future release of the USB2230 firmware.

xD Devices:

Issue:	Workaround:	Status:
No known issues.		

Issues Not Related to Firmware

Issue:	Workaround:	Status:
Due to the write caching functionality of Windows, data corruption can sometimes occur if the media is removed improperly.	Before removing any piece of media, you should right click the drive icon in Windows Explorer and select "Eject" from the context menu. This will force the operating system to perform a write of any cached data.	Limitation of the OS.
Reading or writing multiple media types simultaneously will generally happen at the slowest media rate.	This is a limitation of the OS. If writes to a slow media type like MS are made while reading from a fast media type like CF or SM, then the read will slow to approximately the rate of the write. This is because the OS must process each command separately. It is not a limitation of the firmware.	Limitation of the OS.
If the USB2230 evaluation board does not have a properly programmed serial number, only one drive will appear in Windows Explorer.	Program a unique serial number into the board using the "USBDM" utility.	
Occasionally, surprise removal of the USB cable during writes to any media type under Windows XP, results in the failure of the device to re-enumerate after being reattached.	Reboot the host.	This appears to be a bug in Windows XP. No mass storage class USB devices will enumerate once the host is in this state.
When MSPPro media is inserted and the device is enumerated drive icons won't come up until media is ready to be read. Per MSPPro spec larger media could takes 10 seconds to be ready	None.	
SMSC IRDA driver functionality is currently only supported in Windows XP	Use a host computer running Windows XP if IRDA functionality is needed.	NA
Benchmarks taken when a Tapwave Zodiac1 handheld is transferring files via IrDA at 1.1 speeds to the USB2230 vary from several seconds up to four minutes depending on the specific host computer used. (Files transferred from the USB2230 to the Zodiac1 pass benchmark testing)	Connect the USB2230 to a USB 2.0 host when transferring files from the Zodiac1.	Currently under investigation. May be fixed in a future release of the USB2230 firmware
Some host systems will not communicate via IrDA using the USB2230 or any third party IrDA device.	Right click on "My Computer", select "Manage", under "Services and Application" double click on "Services", highlight "Infrared Monitor", and select "Restart Service"	This is a limitation of certain hosts. It is not seen in many hosts, but those that have this problem do not work with any IrDA devices without completing the work around steps.
Setting attribute byte 3, bit 1 to one may result in USB2230 not functioning properly.	Set attribute byte 3, bit 1 to zero as specified in the attribute bit definitions section.	It is an invalid configuration to set this bit; the next release will prevent a user from being able to set this.
Some Memory Sticks can not be formatted with the Sony Format utility if "Physical Device Display" is not checked	Check the "Physical Device Display" if the MS does not have the option to format. After the first format, this box may not need to be checked for future formats.	This is a limitation of the Sony Format tool, and is mentioned in the applications help topics.
IRDA transmit icon intermittently remains on the desktop when there is no IRDA device in range	Reboot the host	This appears to be a limitation of the OS's IrDA stack.