



STANDARD
MICROSYSTEMS
CORPORATION

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

USB97C210 Software Release Notes

Version 0.0.0.250

Updated 12-22-03

The information contained herein is confidential, is submitted in confidence, and is proprietary information of Standard Microsystems Corporation, and shall only be used in the furtherance of the agreement of which this document forms a part, and shall not, without Standard Microsystems Corporation's prior written approval, be reproduced or furnished to others. The information contained herein may not be disclosed to a third party without the consent of Standard Microsystems Corporation, and then, only pursuant to a Standard Microsystems Corporation approved non-disclosure agreement.

Standard Microsystems Corporation assumes no liability for incidental or consequential damages arising from the use of the information contained herein, and reserves the right to update, revise, or change any information in this document without notice.

Table of Contents

Firmware Version History.....	4
Firmware Version History (cont.).....	5
Hardware Errata.....	8
The Non-Volatile Store Data	10
Using Flash ROM to Store the NVStore Data	10
Using the DFUTest Sample Application.....	10
Using the DFUTest DAT Editor.....	11
Attribute Bit Definitions	11
The Attributes Calculator	13
Programming the NVStore Data.....	13
Programming the NVStore Data.....	14
LUN Configuration and Icon Sharing.....	16
LUN Configuration	16
Icon Sharing	16
Using Device Firmware Upgrade (DFU)	17
Overview	17
Files Required for DFU	17
Creating the 128KB DFU Capable Flash Binary “both.bin”	18
Preparing a Device for DFU Operation.....	19
Choosing a Flash Eeprom for Your Device.....	19
Setting up the Hardware	19
Performing a Firmware Upgrade with the DFUTest Application.....	20
Creating a DFU Uploadable File	21
Using the DFU.exe Utility.....	21
Using the DFU.exe Utility.....	22
Building a DFU Application	23
Driver Overview	23
The SMSC DFU API.....	24
Using the USB97C210 Custom Icons Package.....	42
Contents of the USB97C210 Custom Icons Package	42
Creating the Required SetIcon Ini Files.....	42
Manually Installing the Custom Icons Application Files	44
A Sample Ini File	44
Creating a Windows Installer for the Custom Icons Application Files	45
Troubleshooting the Custom Icons Application	45
Using the Production Line Descriptor Update Utility (PLDU)	46
Creating the PLDU ini File.....	46
A Sample PLDU ini File	47
Setting Up the PLDU Application.....	48
Using the PLDU to Update Device Descriptors	48
Using the PLDU to Update Firmware	49
Using the Production Line Test Utility (PLTU).....	51
Creating the PLTU ini File.....	51
A Sample PLTU ini File.....	52
Setting Up the PLTU Application	53
Using the PLTU to Test Multiple Devices	53
Using the QuickTest Production Line Read/Write Test Utility	54
Known Issues with the USB97C210 Production Line Utilities.....	55
Using the EPRMUPDT.exe Utility.....	56
Using the CheckROM.exe Utility.....	58
Using the MSPro Format Utility.....	59
Using the Windows XP Special Memory Stick Format Registry Key	60
Using the KillReg Utility.....	61
Using the Swapdrv Utility.....	62
Using the USB97C210 with Linux	63
Media Tested with the USB97C210	64

USB97C210 Performance Benchmarks.....	65
GPIO Assignment Table.....	66
Known USB97C210 Firmware Related Issues.....	67
General:	67
CF Devices:	67
MS Devices:	67
SM Devices:	67
SD/MMC Devices:	67
xD Devices:	67
Issues Not Related to USB97C210 Firmware	68

Firmware Version History

For firmware revision history prior to version 0.0.0.96, refer to the USB97C210 release notes for version 0.0.0.125 dated November 12th, 2002.

- Version 0.0.0.96: External Evaluation Build. Fixed the compatibility bug where certain CF cards (e-data) could not be read by the USB97C210.
Fixed the bug causing data corruption during writes to certain Memory Stick cards.
Changed the firmware to allow elimination of the eeprom by writing NVStore data directly to the flash. Currently only the SST39VF010 chip is supported for this functionality.
The firmware is now distributed in four different versions for each LUN configuration, one for devices that use an EEPROM, one for devices that do not use an EEPROM, one with suspend enabled, and one with suspend disabled. The format of the EEPROM.DAT file has changed. Previously programmed eeproms must be reprogrammed with the new data format in order to function correctly with this version of the firmware.
Added code to allow customization of the GPIO 0 LED blink behavior by setting 2 bytes in the eeprom.dat data.
Added code to allow programming of the bmAttributes and MaxPower descriptors in the NVStore area, by setting 2 bytes in the eeprom.dat data.
Added code to allow programming of the LUN device IDs in the nvstore area.
Added additional functionality via the attributes bytes in the eeprom.dat data.
Modified the way the device responds to SCSI Inquiry commands.
- Version 0.0.0.109: External Evaluation Build.
Adjusted the Secure Digital timeout to accommodate slower SD cards.
Fixed the bug causing the 210 to not suspend properly. This eliminates the need for SUSPEND ON and SUSPEND OFF versions of the firmware.
Modified the Write210 application to use the new eeprom.dat signature, and preserve all of the Write210 non-editable bytes.
Fixed bug in the descriptors which caused the wrong USB revision to be reported.
Fixed bug which may have caused data corruption during writes to Smart Media cards.
Fixed bug causing the WHQL USB Manual Interoperability test to fail.
Fixed bug causing the activity LED to remain lit after the device is suspended.
Added code to implement SM 1 bit ECC correction at high speed.
- Version 0.0.0.116: External Evaluation Build.
Added two new fields to the NVStore data area, "Inquiry Manufacturer ID String" and "Inquiry Product ID String." These two fields can be used in place of the USB descriptors, "Manufacturer ID String" and "Product ID String" by setting the attribute bits (See the section on programming the NVStore data).
Released a new MacOS 8.6 and 9.x mass storage class driver with multi-LUN support.
Fixes bug where single bit error correction was not reporting an error at full speed.
- Version 0.0.0.125: External Evaluation Build.
Added additional functionality to control the state of the activity LED during suspend.
Fixes a which caused corruption of Memory Stick media under certain conditions.
Improves 1 and 2-bit error handling at high speed.
Fixes a bug which caused the 210 to sometimes not enumerate properly on OHCI host controllers.
Fixes a bug which caused SD overruns on some systems.
Released version 2.10 of the Win2K MSC driver which fixes the bug causing device IO errors after resuming from suspend/hibernate with the SetIcon application installed.
Released a new version of the MacOS 8.6-9.2 MSC driver which features custom icon replacement capability.
- Version 0.0.0.129: External Evaluation Build.
Fixed the Smart Media SSFDC compliance issue for block address error correction.

Firmware Version History (cont.)

Version 0.0.0.160: External Evaluation Build.

- Added LUN configuration capability by setting bytes in the non-volatile store data.
- Added Icon Sharing capability to allow more than one media type to share a common icon. Typical applications for icon sharing would be devices with multi-card adapters.
- Added a bit in the attribute bytes to turn off Smart Media CIS checking. This will allow the USB97C210 to work with non-compliant Smart Media cards.
- Added support for MS-Pro media. (Serial compatibility mode).
- Modified the Windows 2000 driver (v2.30) to only display one entry in the usb system tray left click safe removal dialog.
- Fixed a bug causing the Kingmax 8MB and Lexar 512MB (24x) Compact Flash cards to not work properly with the USB97C210.
- Fixed a bug in the MacOS 8.6-9.2 driver which caused extra icons to be displayed when using the Icon Sharing functionality of the 210. If you are using Icon Sharing, you should use the updated MacOS 9 driver.
- Modified the SetIcon application to use a new architecture. The changes correct the problems users were experiencing with icons not appearing in Windows XP (SP1).

Version 0.0.0.163: External Evaluation Build.

- Fixed a bug which caused writes to IBM Microdrives to be extremely slow.
- Added a bit in the NVStore Attribute data to turn on Compact Flash compatibility mode. This will force CF cards to operate in slow PIO-0 mode only. Turning this feature on will lower the performance of CF cards, but may allow some non-compliant cards to work with the USB97C210.
- Modified the schematics in the release notes to show the proper hardware implementation for using GPIO5 as an SD card insert indicator.

Version 0.0.0.197: External Evaluation Build.

Firmware:

- Fixed a bug causing MMC transfers to fail if the host was suspended at any time during the transfer.
- Modified the behavior of GPIO7 to drive an LED by going high when the device is in an unconfigured or suspended state.
- Fixed a bug causing the device to become unresponsive if MS Pro media was removed during a format operation.
- Fixed the delay time in waiting for the clock to stabilize. It was in the range of 48.3 - 298ms, but has been corrected to be from 7.2 to 29ms.
- Modified the function of the code such that when VBus is absent or the device is suspended, all flash interfaces are un-powered and set to a high impedance state. This is for battery powered devices, where the app needs to relinquish control of the flash interface and go to sleep, while the kernel is still not suspended.
- Fixed a bug which caused the 210 to become unresponsive when used with certain hosts after several reboots.
- Fixed a bug which caused the media activity LEDs to not initialize properly after resuming from suspend.

Applications:

- Modified the KillReg utility to accept more than a single PID in its ini file.
- Included a Windows 98 safe removal utility (98SafeRemove.exe) that detects the plug / unplug of SMSC USB Mass Storage Devices that utilize the SMSC Windows 98 MSC driver.
- Included a new streamline version of the Production Line Test Utility called "QuickTest". QuickTest is substantially faster than the PLTU because it uses the SMSC filter driver to bypass the Windows file system. The test performs quick read/write tests of up to (4) USB97C210 devices at a time in a production line test environment.
- Fixed a bug in the Windows 2000 multi-LUN mass storage class driver which caused a blue screen after a reboot when used with OMI's EHCI drivers.

0.0.0.216: External Evaluation Build.

Firmware:

- Modified the firmware to improve the transfer speeds of certain high-end Lexar Compact Flash cards.
- Fixed a bug in the Compact Flash media identification code, related to identifying the media supported PIO mode of operation and setting the right mode on the host.
- Fixed a bug with Memory Stick media surprise removal during a read or write.
- Fixed a bug with Secure Digital media surprise removal during a read or write.
- Modified Memory Stick 2 bit ECC error checking to work properly with non-compliant cards.
- Fixed a bug which caused the activity LED to come on and remain lit when power was applied to a self-powered 223 device, while the USB cable was detached.
- Fixed a bug which caused the unconfigured LED (GPIO7) to not function correctly under Windows 98 with certain EHCI drivers.
- Fixed a bug which caused the Secure Digital LED to flash briefly during enumeration.
- Added support for Sony High Speed Memory Stick.
- Added support for xD Picture Cards.
- Added support for the Sony Memory Stick Format Application.

Applications:

- Modified the SetIcon utility (v1.2.0.6) to fix a bug which caused a “No disk in drive” error message to appear in Windows XP SP1 under certain conditions.
- Modified the Windows 98 Safe Removal Utility (v1.0.0.4) to display more descriptive error messages when an error occurs while stopping the device.
- Modified the Attributes Calculator Utility (v.08) to allow both encoding and decoding of attribute values. Please note that you must have the Microsoft Dot Net framework installed on your PC in order to run the Attributes Calculator utility.
- Fixed a bug in the SMSC FormatPro utility (1.0.0.4) which prevented it from recognizing multiple devices in Windows 98 and Me.
- Added a Japanese version of the SMSC FormatPro utility to the application software distribution package.

0.0.0.250: External Evaluation Build

Firmware:

- Fixed issue with using write protected SD cards in a CF to SD adapter.
- Added support for xD picture card 9A command for compliance to the xD specification.
Note: This feature is available only when GPIO4 is used to identify the xD card. The xD card socket must have two active-low card detect (CD) signals. xD uses the Smart Media (SM) interface and one CD is connected to the SM_CD# pin of USB97C210 to identify card insertion/extraction. The other CD signal is connected to GPIO4 via a 10K resistor to distinguish the xD card from the SM card. A 330K pull-up is also required at this CD pin. GPIO4 is now shared for EEPROM EE_DIO and xD card identification.
- Smart Media 1 bit ECC errors were not handled properly depending on location in block. This has been corrected in this release.
- Fixed a bug which prevented the 210 from reading certain Memory Stick cards that contained block errors.

0.0.0.250: External Evaluation Build. (Continued)

- Additional mapper code changes to improve compliance and reliability were added as below:
 - Fixed an issue in Memory Stick so that when the device starts writing from the first sector of a block (e.g., xx00H, xx20H, xx40H), the Update Status of the destination physical block remains set.
 - The exit condition on the search for alternate blocks, for Memory Stick was incorrect and is now fixed.
 - Fixed 1 and 2 bit ECC error on the second or later page in Memory Stick.
 - Fixed issue on Memory Stick such that if an ECC error was on the last page of a split, but not the last page of a transfer, this would cause the `fmc_xfer` to become unresponsive, until the host issued a USB reset.
 - Changed firmware to determine write protect status before attempting to pre-erase blank blocks when building a zone map. This caused Memory Stick to report "CRC Error" on media even after the write protect switch was moved to the unlocked position if Memory Stick media was originally inserted with the media locked.
- Added an attribute bit (Byte 2, bit 5) to make GPIO1 function as a common media LED.
- Added an attribute bit (Byte 2, bit 6) to perform full speed 1-bit ECC error correction for Smart Media cards in software. Previously the 210 was unable to correct 1-bit ECC errors during full speed operation. Please be aware that setting this bit will result in about a 50% performance drop for Smart Media transfers because of the processor overhead involved in doing ECC checking.
- Added a new attribute bit (Byte 2, bit 7) to allow the device to skip the status byte check in the extra data area for Smart Media cards. This speeds up the map rebuilding process which corrects the hiccups in MPEG playback on SM cards. Warning: Setting this bit makes the device non-compliant with the Smart Media specification.
- Fixed a bug which caused LUN ID strings to be misreported. This only occurred under certain LUN configuration and icon sharing schemes when media was inserted at the time of enumeration.
- Fixed a bug in self powered operation which caused the activity LED to stay on when D+ and D- were disconnected but VBus was present.
- Added support for ST M93C66-W EEPROM and improved descriptor update time.
- Added a small delay to the enumeration time for NO.EEPROM versions of the firmware to correct an issue where the device could enumerate yellow banded under certain configurations behind full speed hubs.
- Fixed a bug which caused NVStore updates to occasionally fail with the NO EEPROM version of the firmware.

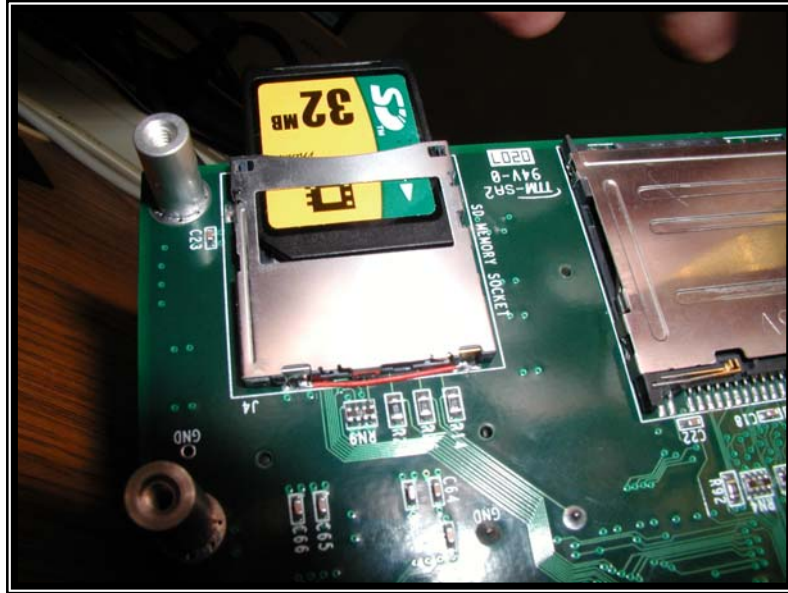
Applications:

- Modified the QuickTest application (v1.0.0.3) to include a "Stop Test" button which allows the user to cancel any tests in progress.
- Modified the software installer to provide multi-language support. The included software installer now supports the following languages: English, Chinese, Danish, Dutch, French, German, Italian, Japanese, Korean, Polish, Russian, Spanish, and Swedish.
- Added two new DOS utilities: `EPRMUPDT.exe` which is used to program the NVStore, and `CheckROM.exe` which is used to verify the firmware revision and check the validity of the NVStore data.
- Included an updated version of the Attributes Calculator (v.11) which adds definitions for the new attribute bits.
- Modified the KillReg utility (v1.0.0.4) to work with all operating systems. Now an OEM can call KillReg at the beginning of an installation to eliminate residual device entries from the registry.

Hardware Errata

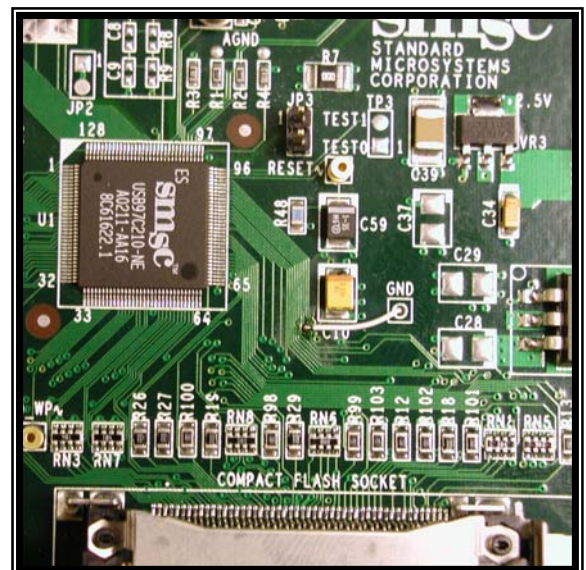
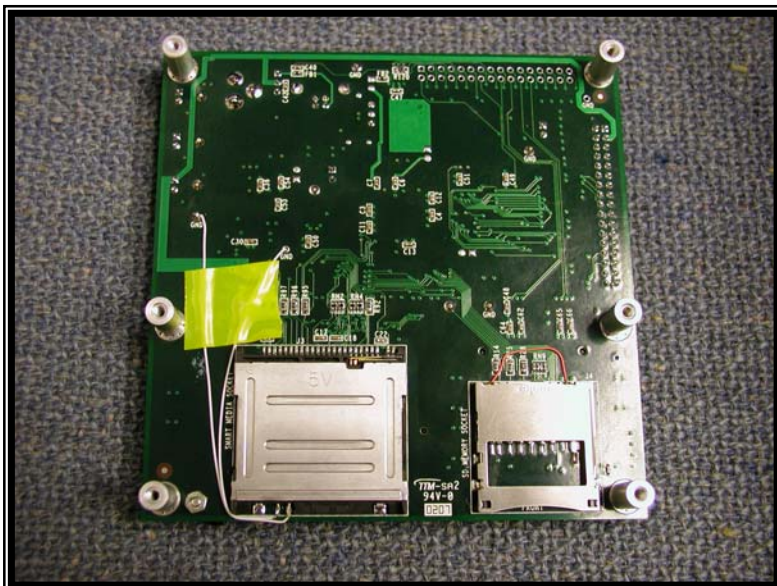
SD Modification: The following modifications must be made to the USB97C210 Evaluation Board Assy 6238 Rev A. (if the board you have does not already have them) in order to ensure compliance with the USB spec, and proper operation of the firmware. Rev. B boards already have this modification and need not be changed.

1. Add a jumper wire on the backside of the SD connector. A wire (see red wire in photo) is added between the two large tabs on the back of the connector bypassing the insert switch.
2. Remove resistors R12, R13, R98, R99, and R100.



Smart Media Write Protect Modification: Some of the USB97C210 evaluation boards do not have the write protect detection switch hooked up. To enable the switch and provide for Smart Media write protect detection, the following hardware modification must be made.

1. Attach a wire from either one of the write protect lines on the front of the Smart Media slot, to a ground on the board.
2. Attach a 2nd wire from the other write protect line, to the SM_nWPS (pin 91) line coming from the chip. (See Photo)



SD Modification (GPIO 5 Used as SD Card Detect): By setting bit 2 of byte 1, GPIO5 can be made to function as an SD card detect pin. SMSC customer evaluation boards will have to be modified as shown below to utilize this feature. The first diagram below shows the SD_WP~ pin connected to a switch on the side of the SD socket which we call the WP Switch. This input is pulled high when a non write protected card is inserted. A pull down resistor causes this pin to go low when a write protected card is inserted (switch is open). A switch commonly available on SD sockets which we call the CD Switch is closed when a card is inserted. Since the firmware is designed to detect a low as a card detect signal on GPIO5 an inverter is used as shown below.



The Non-Volatile Store Data

The Non-Volatile Store contains user modifiable data that is 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, and other modifiable bytes which customize the operation of the firmware.

The NVStore data is programmed in the device by using a text file “EEPROM.DAT”, which is modified and then written to either the EEPROM, or directly into the FlashROM if no EEPROM is present. (Currently, the only Flash ROM supported for NVStore programming is the SST39VF010.) A complete list of the user modifiable data in the EEPROM.DAT file is included in this document. (See the section entitled “Sample EEPROM.DAT File”)

SMSC provides two utilities, the DFUTest application, and the Production Line Test Utility, both of which are capable of programming the NVStore data over the USB. Both applications also contain DAT editors that allow you to create the eeprom.dat file.

Using Flash ROM to Store the NVStore Data

As a cost reduction measure, you can eliminate the need for a serial eeprom in your device by using the SST39VF010 Flash ROM, and the “NO EEPROM” version of the USB97C210 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.

Using the DFUTest Sample Application

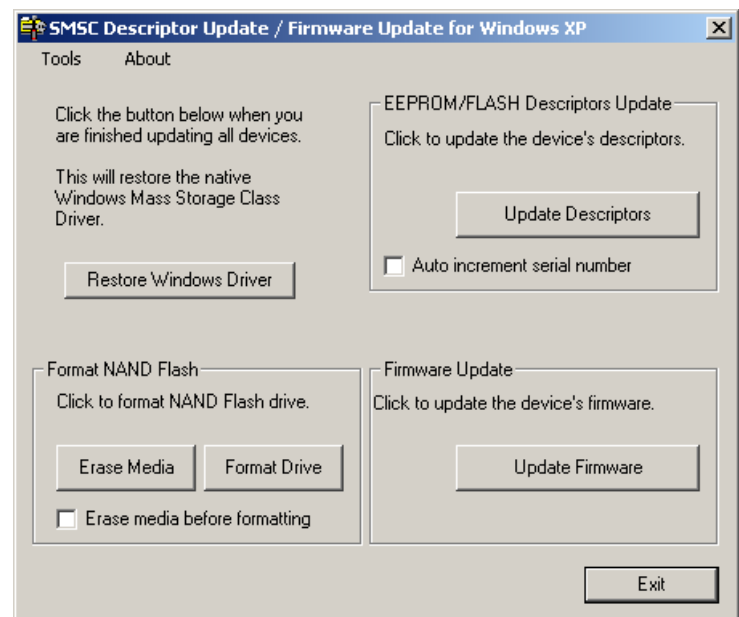
To use the DFUTest application to create the eeprom.dat file, and program the USB97C210 device, the following files are required:

1. The DFU application- (dfuTest.exe)
2. The driver library- (drvlib.dll)
3. The DFU driver- (smscdfu.sys)
4. The DFU installation inf- (smscdfu.inf)
5. (optional) The DFU uploadable firmware- (fmc.dfu) For DFU firmware upgrade only.

Before using the DFUTest application, you must add a VID/PID entry for your device to the “smscdfu.inf” file. This is required for the DFU driver swap to occur properly.

To start the dfuTest application, simply double click the “dfuTest.exe” executable. Once the application starts, you will see the user interface on the right.

To create the eeprom.dat file which will contain the data to be programmed in the non-volatile store area, select “Tools” > “DAT Editor” > “New File” from the application menu.



Using the DFUTest DAT Editor

When creating a new eeprom.dat file in the DFUTest DAT Editor, the dialog to the right appears. All fields should be filled out completely, and the file should be saved using the “Save” or “Save As” buttons.

Attribute Bit Definitions

1. **VID- Vendor ID (2 bytes):** Unique for every vendor. Assigned by the USB Implementers Forum.
2. **PID- Product ID (2 bytes):** Unique to the product. Assigned by the Vendor.
3. **Attributes (4 bytes):** Only the 1st and 2nd Bytes are used. The correct attribute value for your device can be determined using the “Attributes Calculator” utility provided by SMSC. The bit definitions are as follows:

Byte 1, bit 0: Smart Media Timing (Not Used for the USB97C210)

- 1 - NAND flash chips will use the slower, smart media compatible r/w cycle time. This is the recommended setting.
- 0 - NAND flash chips will use the faster 50ns r/w cycle timing for chips that are capable.

Byte 1, bit 1: Enumerate as Hard Drive or Removable Media (Not Used for the USB97C210)

- 1 (default) - NAND flash hard drives always enumerate as removable media.
- 0 - NAND flash hard drives enumerate as removable disks when write protected, and as fixed disks when not write protected.

Byte 1, bit 2: Behavior of GPIO 5

- 1 - Use GPIO5 as an SD card insert indicator.
- 0 (default) - Use GPIO 5 as a High Speed indicator.

Byte 1, bit 3: Behavior of iSerial byte in device descriptor

- 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 the state of the activity LED when suspended, regardless of its idle state.

- 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 - SD cards will be write protected when SW_nWP is high, and writable when SW_nWP is low
- 0 (default) - 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: Smart Media CIS Checking

- 1 - Ignore CIS check for Smart Media to allow the USB97C210 to work with non-compliant cards.
- 0 (default) - Enforce Strict CIS checking for Smart Media cards.

Byte 2, bit 1: Idle processing (Not Used for the USB97C210)

- 1 - idle processing on. Device will perform required erase operations while idle.
- 0 (default) - idle processing off. Device will wait until a write is received before doing required erase operations.

Byte 2, bit 2: Compact Flash Compatibility Mode (Note: This bit should no longer be used. It was originally added to allow compatibility with off brand Compact Flash cards that the 210 was misidentifying. This bug has subsequently been fixed, eliminating the need to use this compatibility bit.)

- 1 - Compact Flash will operate in slow PIO-0 mode only.
- 0 (default) - Compact Flash will operate at the fastest mode the card reports it can support.

Byte 2, bit 3: Change the Device Response to a Get Status Command (Not Used for the USB97C210)

- 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: Change the USB Version the Device Reports to the Host (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.

DAT Editor - EEPROM.DAT

File

DAT Editor

VID: 0x0424 PID: 0x20FC Attributes: 0x00000000

Language ID: 0x0409 Serial No: 0x777777777777

Manufacturer String: SMSC

Product String: USB 2 Flash Media Device

Format Signature: ata1

Fields For Format "ata1"

bMaxPower: 0x30 Blink Interval: 0x05 Blink Duration: 0x02

bmAttrib: 80 Lun0 ID: CF Lun1 ID: MS

Lun2 ID: SM Lun3 ID: SD/MMC

Inquiry Manufacturers ID String: SMSC

Inquiry Product ID String: USB 2

Number of Icons to Display: 04 CF Lun #: 0x00 MS Lun #: 0x01

SD/MMC #: 0x02 SM Lun #: 0x03

NAND Profile: 0xFFFF NAND HD #: 0xFF

Save Save As Exit

Attribute Bit Definitions (cont.)**Byte 2, bit 6: Perform Software 1-bit ECC Error Correction on Smart Media.**

1 – The device will perform Full Speed 1-bit ECC error correction in software for Smart Media transfers. Please be warned that setting this bit will result in approx. 50% transfer performance drop for Smart Media due to the processor overhead required to do ECC checking in software.

0(default) – The device will not correct 1-bit ECC errors during full speed Smart Media transfers.

Byte 2, bit 7: Bypass Status Byte Check for Smart Media Cards.

1 – The device will bypass the Smart Media status byte check in the extra data area, speeding up the map building process. Caution: Setting this bit makes the design noncompliant to the Smart Media specification.

0(default) – The device will not bypass the Smart Media status byte check. (Smart Media spec compliant).

All other bits and bytes are reserved and should be set to 0.

4. **Language ID (2 bytes):** 0409 is the Language Code for English. Other language codes may be found in the USB 2.0 specification.
5. **Serial Number (12 Hex Digits Max):** Unique to each device. The serial number can be up to 12 hex digits, written in the eeprom.dat file as unicode.
6. **Manufacturers String (28 Characters Max):** Used to hold a descriptive manufacturer string.
7. **Product ID String (28 Characters Max):** Used to hold a string to identify the product. The user will see this string during the USB enumeration process in Windows.
8. **Format Signature-** Do not change. For the USB97C210, this should remain “ata1”.
9. **bmAttributes (1 byte)-** Per USB Specification.
 - 80 – Device is bus powered.
 - C0 – Device is self powered.
10. **MaxPower (1 byte)-** Per USB Specification. Do not set this value greater than 100mA.
 - 01 – 2mA
 - 31 – 98mA
11. **GPIO 0/1 LED 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 10ms.
12. **GPIO 0/1 Blink After Access Time (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.
13. **LUN ID Strings (7 bytes each)-** There are four LUN ID strings corresponding to LUNs 0,1,2 and 3.
14. **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.
15. **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. (See the section of this document entitled “LUN Configuration and Icon Sharing.”)
16. **NAND Profile (2 Bytes): (Not Used for the USB97C210)** This is where the NAND performance profile is specified for controllers that use them.

The Attributes Calculator

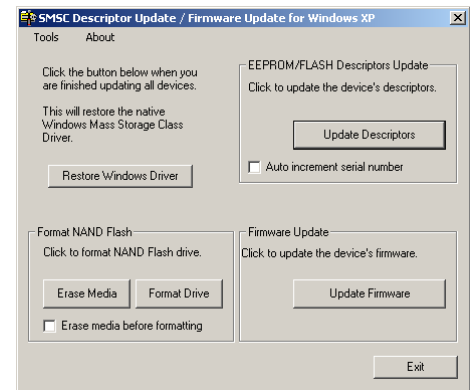
SMSC provides a small utility called the Attributes Calculator which can be used to calculate the attribute values for your device. **In order to run the utility, you must have the latest Microsoft NET framework installed on your PC.** The NET framework can be obtained through a normal Windows Update, or you can download it manually from the Microsoft website at: <http://msdn.microsoft.com/netframework/downloads/howtoget.aspx> . To use the utility, simply select each of the Attribute byte tabs and check the boxes for the bits you want to use. The attribute bytes are calculated in real-time and displayed at the top of the application. If you hover the mouse pointer over any of the bits, a complete definition and option summary is displayed on the right.

The screenshot shows the 'Attributes Calculator v.08' application window. It features a title bar with standard Windows window controls. The main interface is divided into several sections:

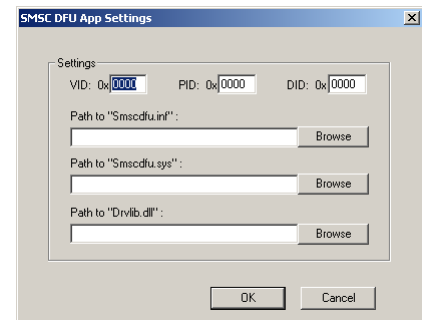
- Attribute Byte Values:** A section with four input fields labeled 'Byte 1', 'Byte 2', 'Byte 3', and 'Byte 4', each containing the value '00'.
- Attribute Hex Value:** A section displaying the calculated hex value '0x00000000' in blue text.
- Attribute Byte Tabs:** Four tabs labeled 'Attribute Byte 1', 'Attribute Byte 2', 'Attribute Byte 3', and 'Attribute Byte 4'. The 'Attribute Byte 1' tab is currently selected.
- Attribute Byte 1 Options:** A list of eight checkboxes under the 'Attribute Byte 1' tab, all of which are currently unchecked:
 - ☐ Use Slow NAND Flash Media Timing
 - ☐ Enumerate NAND device as Removable Media
 - ☐ Use GPIO5 as an SD Card Insert Indicator
 - ☐ Report iSerial Byte as Zero in Device Descriptor
 - ☐ Use Inquiry Manufacturer and Product ID Strings
 - ☐ Set the Activity LED GPIO to High When Suspended
 - ☐ Reverse SD Card Write Protect Sense
 - ☐ Make SD Cards Write Protected Always
- Filter by Chip:** A dropdown menu in the top right corner, currently set to 'All'.
- Options:** A large empty rectangular area at the bottom right, intended for displaying detailed information when a specific attribute bit is hovered over.

Programming the NVStore Data

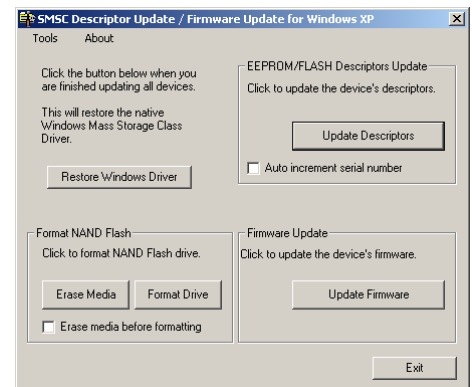
Once the eeprom.dat file has been created with the DFUTest application, you are ready to program the NVStore data into your device.



Once the eeprom.dat file is complete, press the “Update Descriptors” button on the DFUTest application to program the NVStore data. A dialog like the one on the right will appear. Enter the current VID/PID/DID for the device, browse to the path of each of the three required files, and select “OK”. The operation will report completion once the data has been programmed. Note that you can also use the USB97C210 Production Line Test Utility to program and configure multiple 210 devices at once. See the section of this document entitled “Using the Production Line Test Utility.”



The DFUTest application can also be used to upgrade the firmware for the USB97C210 device by pressing the “Update Firmware” button. For complete information on performing a device firmware upgrade (DFU) operation, see the section of this document entitled “Using Device Firmware Upgrade.”



Sample EEPROM.DAT File

Below is an example of the contents of the “EEPROM.DAT” file, displayed here in columnar format for clarity’s sake. For each of the string descriptors, the first byte is the length of the descriptor including the length byte itself. The next byte is the “03” String ID, followed by the string itself. For example, the string “SMSC” would be “0A 03 53 00 4D 00 53 00 43 00 00 00”. Note that “0A” is the length, followed by “03” the String ID, and then the SMSC string in Unicode, terminated with the NULL character “00”.

1A // length of serial	00	80 // bmAttribute - 0xC0 (Self) or 0x80 (Bus) only
03 // string descriptor type	00	30 // bMaxPower - 1 <= bMaxPower <= 0x31
30 // unicode serial number string descriptor	00	05 // GPIO 0 LED blink interval in mult. of 10msec - hi bit indicates idle state: 0-off, 1-on
00	00	05 // number of seconds to keep GPIO 0 LED blinking after access
30	00	43 // ascii, not null terminated, logical lun 0 id string
00	00	46 //CF
30	00	00
00	00	00
30	00	00
00	26 // unicode product string descriptor	00
30	03	00
00	55	00
30	00	4D // ascii, not null terminated, logical lun 1 id string
00	53	53 //MS
30	00	00
00	42	00
30	00	00
00	20	00
30	00	00
00	32	53 // ascii, not null terminated, logical lun 2 id string
30	00	4D //SM
00	20	00
30	00	00
00	46	00
31	00	00
00	4C	00
43	00	53 // ascii, not null terminated, logical lun 4 id string
00	00	44 //SD/MMC
24 // vid lo	41	2F
04 // vid hi	00	4D
FC // pid lo	53	4D
20 // pid hi	00	43
04 // langid length	48	00
03 // descriptor type: string	00	FF // reserved - used only by SD
09 // langid lo	20	FF
04 // langid hi	00	FF
0A // unicode manufacturer's string descriptor	44	FF
03	00	FF
53	45	FF
00	00	FF
4D	56	FF
00	00	FF
53	49	FF
00	00	FF
43	43	FF
00	00	FF
00	45	FF
00	00	FF
00	00	FF
00	00	FF
00	00	FF
00	00	FF
00	00	FF
00	00	FF
00	00	FF
00	00	FF
00	00	FF
00	00	00 // inquiry manufacturer id string
00	00	00
00	00	00
00	00	00
00	00	00
00	00	00
00	00	00
00	00	00
00	00	00 // inquiry product id string
00	00	00
00	00	00
00	00	00
00	00	00
00	00	FF // max number of luns
00	00 // attributes lo word lo byte	FF // CF lun number
00	00 // attributes lo word hi byte	FF // MS lun number
00	00 // attributes hi word lo byte	FF // SM lun number
00	00 // password	FF // SD/MMC lun number
00	00	FF // NAND lun number
00	00	FF // reserved - to set clock speed - edit manually
00	00	FF // NAND profile hi byte
00	00	FF // NAND profile lo byte
00	00	FF // reserved - fpga, pwr_mgmt flags - edit manually
00	00	FF // reserved
00	00	61 // signature
00	00	74
00	00	61
00	00	31
00	00	

LUN Configuration and Icon Sharing

LUN Configuration

LUN (Logical Unit Number) is the term given to each available media type in the USB97C210. The USB97C210 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 5 bytes in the NVStore data. (See the section entitled “Using the DFUTest DAT Editor”).

Example: The example on the right shows the correct settings for a 210 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”.

The screenshot shows the DAT Editor window with the following settings:

- VID: 0x0424, PID: 0x20FC, Attributes: 0x00000000
- Language ID: 0x0409, Serial No: 0x7777777777
- Manufacturer String: SMSC
- Product String: USB 2 Flash Media Device
- Format Signature: ata1
- Fields For Format "ata1":
 - bMaxPower: 0x30, Blink Interval: 0x05, Blink Duration: 0x02
 - bmAttrib: 80
 - Lun0 ID: CF, Lun1 ID: MS, Lun2 ID: SM, Lun3 ID: SD/MMC
 - Inquiry Manufacturers ID String: SMSC
 - Inquiry Product ID String: USB 2
 - Number of Icons to Display: 03
 - CF Lun #: 0x02, MS Lun #: 0x00, SD/MMC #: 0xFF, SM Lun #: 0x01
 - NAND Profile: 0xFFFF, NAND HD #: 0xFF

Icon Sharing

In addition to LUN configuration, the USB97C210 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 USB97C210 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 USB97C210 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 210 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)

The screenshot shows the DAT Editor window with the following settings:

- VID: 0x0424, PID: 0x20FC, Attributes: 0x00000000
- Language ID: 0x0409, Serial No: 0x7777777777
- Manufacturer String: SMSC
- Product String: USB 2 Flash Media Device
- Format Signature: ata1
- Fields For Format "ata1":
 - bMaxPower: 0x30, Blink Interval: 0x05, Blink Duration: 0x02
 - bmAttrib: 80
 - Lun0 ID: CF, Lun1 ID: MS, Lun2 ID: SM, Lun3 ID: SD/MMC
 - Inquiry Manufacturers ID String: SMSC
 - Inquiry Product ID String: USB 2
 - Number of Icons to Display: 02
 - CF Lun #: 0x00, MS Lun #: 0x01, SD/MMC #: 0x01, SM Lun #: 0x01
 - NAND Profile: 0xFFFF, NAND HD #: 0xFF

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 the chip at the time it is initially programmed. 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.

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, and a DFU driver API which customers can use to quickly develop custom DFU applications. 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

The following files are provided for DFU evaluation:

dfuTest.exe –A sample DFU application which demonstrates the use of the API and the procedure for updating the firmware and NVStore data.

drvlib.dll –A dynamic link library loaded with “smscdfu.sys” which handles all of the non-DFU specific operations such as PNP message handling and basic WDM and USB support.

smscdfu.sys -This is the DFU driver which is loaded prior to performing a firmware or eeprom update operation. It is responsible for handling the DFU specific function calls from the DFU application.

smscdfu.inf –The file responsible for loading the “smscdfu.sys” DFU driver. The contents of this file should never be altered.

eeprom.dat –A text file containing the changeable descriptor information used to update the NVStore. This file can be created and edited in the DAT Editor (under the Tools menu) in the DFUTest application.

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.

dfu.hex -The DFU execution code that is inserted into the lower 64kb of a 128kb flash when it is initially programmed. This hex file is converted to a 64kb binary file with the “hex2bin.exe” utility, and then appended to the 64kb “fmc.bin” file to create the 128kb flash image. (Included with the USB97C210 firmware).

fmc.hex -The USB97C210 device firmware that is inserted into the upper 64kb of a 128kb flash when it is initially programmed. This hex file is converted to a 64kb binary file with the “hex2bin.exe” utility, and then appended to the 64kb “dfu.bin” file to create the 128kb flash image. (Included with the USB97C210 firmware).

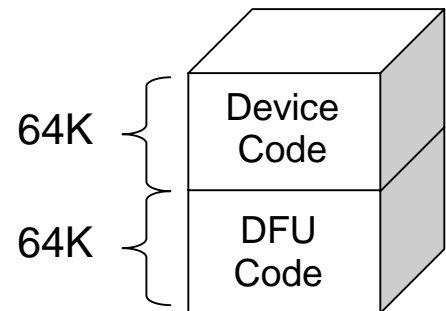
fmc.dfu -A firmware image that can be uploaded to the device. This file is created by the user. This document explains in detail how to make downloadable DFU images through the use of the “DFU.exe” utility, which appends a DFU file suffix to the firmware file to be uploaded to the device. (This file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

Application Source Code -All of the source code for the dfuTest sample application, as well as the DFULIB.LIB link library used to create custom DFU applications.

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 USB97C210 device code. The device code is converted to a 64KB binary file, and appended to the DFU code, which has also been converted to a 64KB binary file. 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 64KB block, and the device code occupies the upper 64KB block.

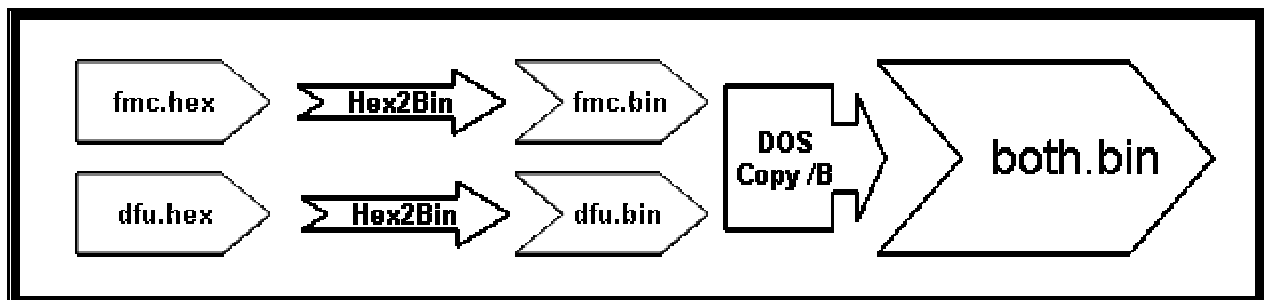
In normal operation, a DFU capable USB97C210 device executes only the device code in the upper 64KB block of memory. This code allows it to function as a normal USB 2.0 flash media controller. However, when the device is switched to DFU mode, the DFU code in the lower 64KB block begins executing and 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 USB97C210 device code in the upper 64KB 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) `dfu.hex` (The DFU code)

The “dfu.hex” file is provided by SMSC, and provides programming support for a limited number of eeproms. The “fmc.hex” file is the standard USB97C210 device firmware. These two files, “dfu.hex” and “fmc.hex,” are both converted to 64KB binary files with the “hex2bin.exe” utility, and then appended to each other with a DOS copy command. 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 -L65536 dfu.hex dfu.bin
hex2bin -L65534 fmc.hex fmc.bin
copy /Y /B dfu.bin /B + fmc.bin /B both.bin /B
```

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 64KB of memory until it is called upon to perform a device firmware upgrade operation.

Once the flash has been programmed with the “both.bin” file, it may be inserted into the 210’s flash socket in preparation for DFU operation.

Choosing a Flash Eeprom for Your Device

SMSC provides customers the “dfu.hex” file which supports a limited number of eeproms commonly used with the USB97C210. At present, the DFU package provides support for the **SST39VF010**, the **MX29F001**, the **AM29LV010B**, the **STM29W010B**, or the **STM29F010B** flash eeproms. 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. Also, it should have equivalent programming characteristics as the three 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 OrangeMicro’s have been found to have flaws 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.

Performing a Firmware Upgrade with the DFUTest Application

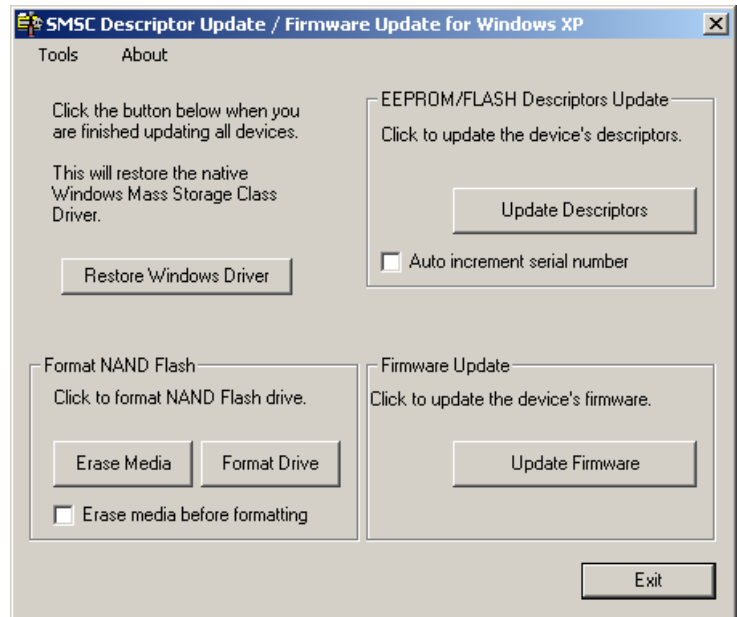
The following files are required in order to perform a device firmware upgrade:

1. The DFU application- (DFUTest.exe)
2. The driver library- (drvlib.dll)
3. The DFU driver- (smscdfu.sys)
4. The DFU installation inf- (smscdfu.inf)
5. The updated firmware image- (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. (See the section of this document entitled "The Non-Volatile Store Area."

Before using the DFUTest application, you must add a VID/PID entry for your device to the "smscdfu.inf" file. This is required for the DFU driver swap to occur properly.

To start the dfuTest210 application, simply double click the "dfuTest.exe" executable. Once the application starts, you will see the user interface on the right. Pressing either the "Update Descriptors" or "Update Firmware" button causes the DFU driver to load. This driver is required for the update to take place. From a user's perspective, the drive icon(s) will disappear once the device enters DFU mode. The DFU upload process is not completed until the "Operation Complete" dialog appears. The application itself does not provide any indication of the progress of the update. A typical firmware update takes about 1 minute to complete. To unload the DFU driver, press the "Restore Windows Driver" button. This will restore the Windows mass storage class driver, and allow the device to be operated normally. Note: In order for the new descriptor information to appear, you must unplug the device, and then plug it back into the host. On attach, the device will begin using the new data in the NVStore area.

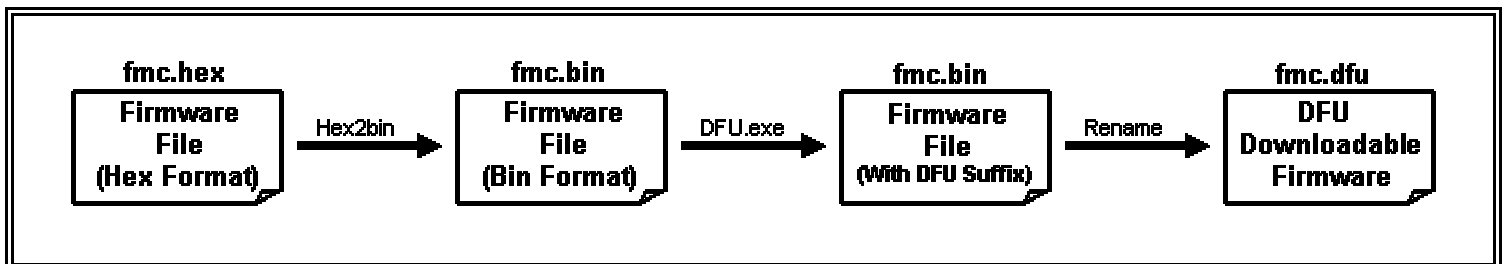


Creating a DFU Uploadable File

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:

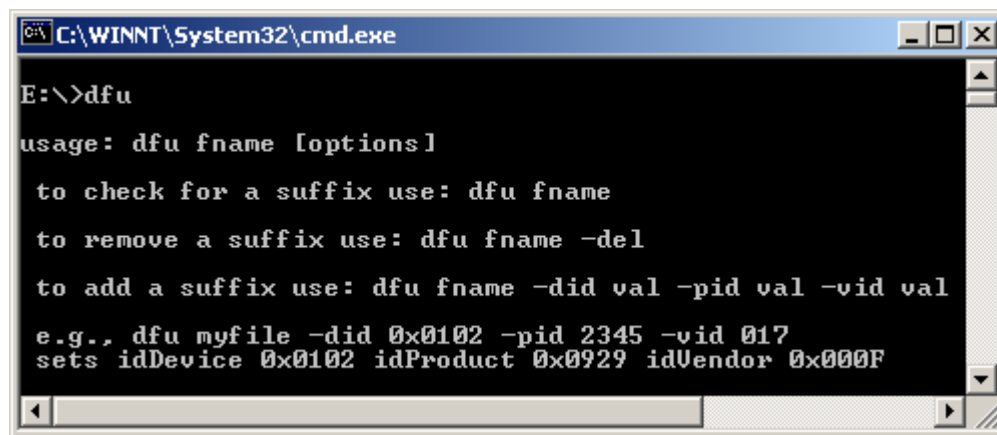
DFU.exe <filename> -del

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 0x1234 -pid 0x20FC -vid 0x0424



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

E:\>dfu

usage: dfu fname [options]

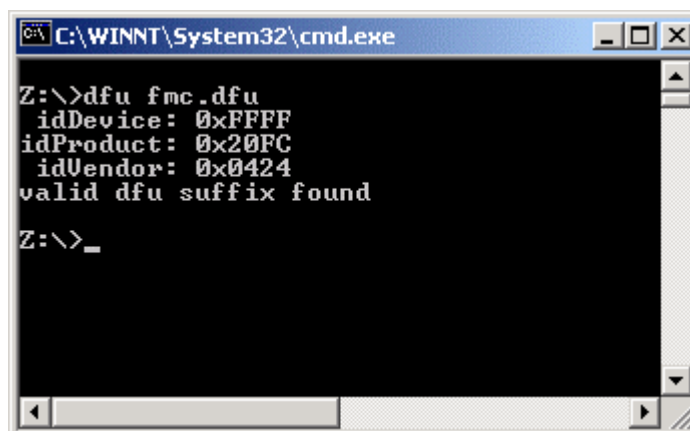
to check for a suffix use: dfu fname

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 dfuTest210 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:\>_
```

Building a DFU Application

SMSC provides the source code for the dfuTest210 sample application, which can be used to template your own custom DFU applications. However, before developing your own application, you should understand the five steps the application must perform to complete the DFU operation:

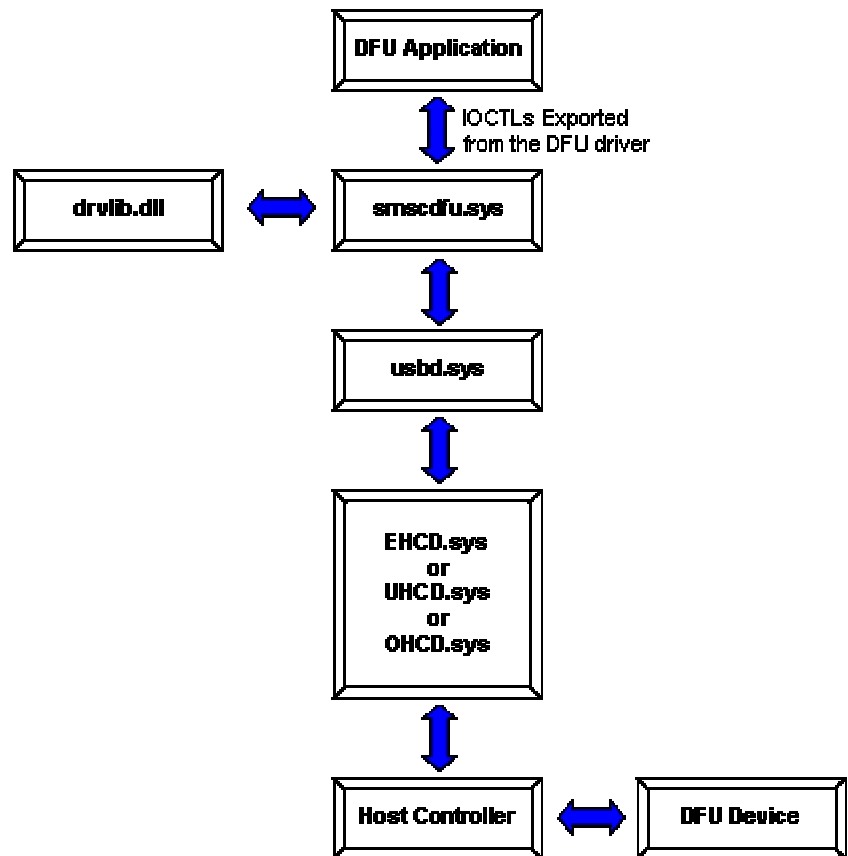
1. Initiate the update
2. Find the device driver attached to the target device's VID/PID
3. Exchange the device's driver with the DFU driver "smscdfu.sys"
4. Perform the update
5. Unload the DFU driver and restore the original device driver.

All of the above steps may be performed through the use of calls to the SMSC DFU API, which is made available to the application when it is linked to the "dfulib.lib" library. A complete list of all the SMSC DFU API function calls, complete with descriptions, usage and commentary is available in Appendix 1 of this document, "The SMSC DFU API".

Driver Overview

The DFU application communicates to the device via IO Control Calls to the DFU driver "smscdfu.sys" as shown in the diagram on the right.

The "smscdfu.sys" driver handles all of the DFU specific requests, while it passes all other requests, such as PNP message handling and USB standard traffic, on to the "drvlib.dll" for handling.



The SMSC DFU API

The following are the list of functions available through the SMSC DFU API, with descriptions, usage, parameters, and commentary describing how they should be implemented in the application. The API is made available to the application by linking to the “dfuLib.lib” library at compile time.

Int32 Start Firmware Update (char* fname, char* infFile, char* sysFile,
char* drvLFile uInt16 vid, uInt16 pid, uInt16 did)

This function allows the updating of the OEM descriptor fields.

Parameters

<i>fname</i>	pointer to a complete path that specifies where the location of the new firmware .bin file resides.
<i>infFile</i>	pointer to a complete path that specifies where the location of the SMSC DFU .inf file resides.
<i>sysFile</i>	pointer to a complete path that specifies where the location of the SMSC DFU .sys driver file resides.
<i>drvLFile</i>	pointer to a complete path that specifies where the location of the SMSC DFU .dll driver lib file resides.
<i>vid</i>	vendor ID of the OEM specific device
<i>pid</i>	product ID of the OEM specific device
<i>did</i>	device ID of the OEM specific device

Comments

The function will install the INF file specified, copying the needed driver files to the Windows System directory. It then initiates a driver swap causing the Windows or OEM specific .sys driver to be replaced by the SMSC DFU driver. The driver swap is done in preparation for the next API call which should follow in sequence. This API call is **Firmware_Update**.

The function returns zero for success, or one of the error codes described in the dfuDLL.h header file.

API Functions**Int32 Firmware Update** (void)

This function allows the updating of the existing application firmware.

Parameters

None

Comments

The function then uses the SMSC DFU driver to initiate a DFU class firmware update, which replaces the existing application firmware with the new firmware. After the firmware is successfully updated, the API call [End_Firmware_Update](#) can be used to restore the original application's device driver allowing normal operation of the device to continue.

The function returns zero for success, or one of the error codes described in the dfuDLL.h header file.

API Functions**Int32 End Firmware Update** (char* originalDriverInfName)

This function terminates the updating of the application firmware and restores the original application device driver.

Parameters

originalDriverInfName pointer to a NULL terminated string that describes the file name only (not path) of the INF file used to enumerate the device in its original application state (i.e, "usbstor.inf")

Comments

Call this function when finished updating all device firmware. This function swaps the DFU driver out of the operating system and restores the original application device driver. You can plug in other devices for update BEFORE calling this function. This function serves as the terminating call to updating all devices. Before calling this function, the DFU is fully installed and used for each device plugged in.

The function returns zero for success, or one of the error codes described in the dfuDLL.h header file.

API Functions

Int32 Start Descriptor Update (char* infFile, char* sysFile, char*
drvLFile uInt16 vid, uInt16 pid, uInt16 did)

This function allows the updating of the OEM descriptor fields.

Parameters

<i>infFile</i>	pointer to a complete path that specifies where the location of the SMSC DFU .inf file resides.
<i>sysFile</i>	pointer to a complete path that specifies where the location of the SMSC DFU .sys driver file resides.
<i>drvLFile</i>	pointer to a complete path that specifies where the location of the SMSC DFU .dll driver lib file resides.
<i>vid</i>	vendor ID of the OEM specific device
<i>pid</i>	product ID of the OEM specific device
<i>did</i>	device ID of the OEM specific device

Comments

The function will install the INF file specified, copying the needed driver files to the Windows System directory. It then initiates a driver swap causing the Windows or OEM specific .sys driver to be replaced by the SMSC DFU driver. The function then uses the SMSC DFU driver to send a vendor specific command to the device firmware, instructing it to rewrite its OEM descriptor table. Upon the next enumeration, the new OEM descriptors will be exported.

The function returns zero for success, or one of the error codes described in the dfuDLL.h header file.

API Functions

Int32 Descriptor Update (char* buffer,uInt32 size)

This function allows the updating of the OEM descriptor fields.

Parameters

<i>buffer</i>	pointer to a 256 byte buffer that contains the formatted OEM data fields to update internal descriptors. This is raw binary data.
<i>size</i>	size of the <i>buffer</i> in bytes (256)

Comments

The function will install the INF file specified, copying the needed driver files to the Windows System directory. It then initiates a driver swap causing the Windows or OEM specific .sys driver to be replaced by the SMSC DFU driver. The function then uses the SMSC DFU driver to send a vendor specific command to the device firmware, instructing it to rewrite its OEM descriptor table. Upon the next enumeration, the new OEM descriptors will be exported.

The function returns zero for success, or one of the error codes described in the dfuDLL.h header file.

API Functions**Int32 End Descriptor Update** (char* originalDriverInfName)

This function terminates the updating of the OEM descriptor fields and restores the application driver.

Parameters

originalDriverInfName pointer to a NULL terminated string that describes the file name only (not path) of the INF file used to enumerate the device in its original application state (i.e, "usbstor.inf")

Comments

Call this function when finished updating all devices. This function swaps the DFU driver out of the operating system and restores the original application device driver. You can plug in other devices to update BEFORE calling this function last. This function serves as the terminating call to updating all devices. Before calling this function, the DFU is fully installed and used for each device plugged in.

The function returns zero for success, or one of the error codes described in the dfuDLL.h header file.

API Functions**Int32 Get Error String** (Int32 errorCode, char* buffer)

This function terminates the updating of the OEM descriptor fields and restores the application driver.

Parameters

errorCode the 32-bit signed error code received from any
DFU library function calls.

buffer a minimum of 512 byte buffer for string
storage.

Comments

Call this function to translate an error code received from the DFU library, into a NULL terminated text string. You must provide 512 bytes of storage for the buffer parameter.

The function returns zero for success, or one of the error codes described in the dfuDLL.h header file.

API Functions

uInt32 Get_OS_Version (char* osString)

This function returns an operating system identification code and string that specifies which platform the DFU library is running on.

Parameters

osString a minimum of 512 byte buffer for string storage.

Comments

Call this function to determine which operating system the DFU library is executing on. This is a utility function that returns a string and code identifier as shown below. See the dfuDLL.h header file for a complete list of operating system codes.

```
#define OS_WINDOWS_95                      0x00
#define OS_WINDOWS_95OSR2                 0x01
#define OS_WINDOWS_NT351                  0x02
#define OS_WINDOWS_98                     0x03
#define OS_WINDOWS_98SE                   0x04
#define OS_WINDOWS_NT40                   0x05
#define OS_WINDOWS_2000                   0x06
#define OS_WINDOWS_XP                     0x07
#define OS_WINDOWS_ME                     0x08
#define OS_WINDOWS_NEWNTOS                0x09
#define OS_WINDOWS_NEWCONSUMEROS         0x0a
```

API Functions**Int32 UpdateFirmware**

```
(char* fname,char* infFile,char* sysFile,
char* drvLFile,uInt16 vid,uInt16 pid,uInt16
did,char* originalDriverInfName);
```

This function allows the updating of the device firmware module.

Parameters

<i>fname</i>	pointer to a complete path that specifies where the location of the new firmware .bin file resides.
<i>infFile</i>	pointer to a complete path that specifies where the location of the SMSC DFU .inf file resides.
<i>sysFile</i>	pointer to a complete path that specifies where the location of the SMSC DFU .sys driver file resides.
<i>drvLFile</i>	pointer to a complete path that specifies where the location of the SMSC DFU .dll driver lib file resides.
<i>vid</i>	vendor ID of the OEM specific device
<i>pid</i>	product ID of the OEM specific device
<i>did</i>	device ID of the OEM specific device
<i>originalDriverInfName</i>	pointer to a NULL terminated string that describes the file name only (not path) of the INF file used to enumerate the device in its original application state (i.e, "usbstor.inf")

Comments

The function will install the INF file specified, copying the needed driver files to the Windows System directory. It then initiates a driver swap causing the Windows or OEM specific .sys driver to be replaced by the SMSC DFU driver. The function then uses the SMSC DFU driver to initiate a DFU class firmware update, which replaces the existing application firmware with the new firmware. After the firmware is successfully updated, the operating system is instructed to swap the DFU device driver with the original application's device driver allowing normal operation of the device to continue.

The functions returns zero for success, or one of the error codes described in the dfuDLL.h header file.

API Functions

Int32 Start Format Drive (char* infFile, char* sysFile, char*
drvLFile uInt16 vid, uInt16 pid, uInt16 did)

This function allows swapping the Mass Storage Class driver with SMSCDFU driver.

Parameters

<i>infFile</i>	pointer to a complete path that specifies where the location of the SMSC DFU .inf file resides.
<i>sysFile</i>	pointer to a complete path that specifies where the location of the SMSC DFU .sys driver file resides.
<i>drvLFile</i>	pointer to a complete path that specifies where the location of the SMSC DFU .dll driver lib file resides.
<i>vid</i>	vendor ID of the OEM specific device
<i>pid</i>	product ID of the OEM specific device
<i>did</i>	device ID of the OEM specific device

Comments

The function will install the INF file specified, copying the needed driver files to the Windows System directory. It then initiates a driver swap causing the Windows or OEM specific .sys driver to be replaced by the SMSC DFU driver.

The function returns zero for success, or one of the error codes described in the dfuDLL.h header file.

API Functions**Int32 Format Drive** (uInt8* Label, BOOL ForceMediaErase)

This function does the formatting of NAND Flash Hard disk drives.

Parameters

<i>Label</i>	pointer to a 11 byte buffer that contains the label of the volume. If this parameter is NULL or points to an empty string, then the volume will contain no Label information.
<i>ForceMediaErase</i>	specifies whether the Flash media is to be erased before formatting the drive.

Comments

If necessary, this function will install the INF file specified, copying the needed driver files to the Windows System directory and initiate a driver swap causing the Windows or OEM specific .sys driver to be replaced by the SMSC DFU driver. The function then uses the SMSC DFU driver to send SCSI commands to the device firmware to create a primary DOS partition and format it to a FAT12, FAT16 or FAT32 volume. The FAT type is determined by the capacity of the drive and cannot be specified by the user.

The function returns zero for success, or one of the error codes described in the dfuDLL.h header file.

API Functions**Int32 End Format Drive** (char* originalDriverInfName)

This function terminates the format process and restores the original application driver.

Parameters

originalDriverInfName pointer to a NULL terminated string that describes the file name only (not path) of the INF file used to enumerate the device in its original application state (i.e, "usbstor.inf")

Comments

Call this function when finished formatting all devices. This function swaps the DFU driver out of the operating system and restores the original application device driver. You can plug in other devices to format BEFORE calling this function last. This function serves as the terminating call to formatting all devices. Before calling this function, the DFU is fully installed and used for each device plugged in.

The function returns zero for success, or one of the error codes described in the dfuDLL.h header file.

API Functions

Int32 Start Erase Media (char* infFile, char* sysFile, char*
drvLFile uInt16 vid, uInt16 pid, uInt16 did)

This function allows swapping the Mass Storage Class driver with SMSCDFU driver.

Parameters

<i>infFile</i>	pointer to a complete path that specifies where the location of the SMSC DFU .inf file resides.
<i>sysFile</i>	pointer to a complete path that specifies where the location of the SMSC DFU .sys driver file resides.
<i>drvLFile</i>	pointer to a complete path that specifies where the location of the SMSC DFU .dll driver lib file resides.
<i>vid</i>	vendor ID of the OEM specific device
<i>pid</i>	product ID of the OEM specific device
<i>did</i>	device ID of the OEM specific device

Comments

The function will install the INF file specified, copying the needed driver files to the Windows System directory. It then initiates a driver swap causing the Windows or OEM specific .sys driver to be replaced by the SMSC DFU driver.

The function returns zero for success, or one of the error codes described in the dfuDLL.h header file.

API Functions**Int32 Erase Media** (void)

This function allows the erasing all valid pages of NAND Flash Hard disk drives.

Parameters

None

Comments

If necessary, this function will install the INF file specified, copying the needed driver files to the Windows System directory and initiate a driver swap causing the Windows or OEM specific .sys driver to be replaced by the SMSC DFU driver. The function then uses the SMSC DFU driver to send SCSI commands to the device firmware to erase every valid page on the media, restoring it to an un-written state.

The function returns zero for success, or one of the error codes described in the dfuDLL.h header file.

API Functions**Int32 End Erase Media** (char* originalDriverInfName)

This function terminates the erase process and restores the original application driver.

Parameters

originalDriverInfName pointer to a NULL terminated string that describes the file name only (not path) of the INF file used to enumerate the device in its original application state (i.e, "usbstor.inf")

Comments

Call this function when finished erasing all devices. This function swaps the DFU driver out of the operating system and restores the original application device driver. You can plug in other devices to erase BEFORE calling this function last. This function serves as the terminating call to erasing all devices. Before calling this function, the DFU is fully installed and used for each device plugged in.

The function returns zero for success, or one of the error codes described in the dfuDLL.h header file.

API Functions

Int32 Start Descriptor Read (char* infFile, char* sysFile, char*
drvLFile uInt16 vid, uInt16 pid, uInt16 did)

This function allows swapping the Mass Storage Class driver with SMSCDFU driver.

Parameters

<i>infFile</i>	pointer to a complete path that specifies where the location of the SMSC DFU .inf file resides.
<i>sysFile</i>	pointer to a complete path that specifies where the location of the SMSC DFU .sys driver file resides.
<i>drvLFile</i>	pointer to a complete path that specifies where the location of the SMSC DFU .dll driver lib file resides.
<i>vid</i>	vendor ID of the OEM specific device
<i>pid</i>	product ID of the OEM specific device
<i>did</i>	device ID of the OEM specific device

Comments

The function will install the INF file specified, copying the needed driver files to the Windows System directory. It then initiates a driver swap causing the Windows or OEM specific .sys driver to be replaced by the SMSC DFU driver.

The function returns zero for success, or one of the error codes described in the dfuDLL.h header file.

API Functions

Int32 Descriptor Read (uInt8* buffer, uInt32* size)

This function allows the reading of device's internal descriptors stored in the EEPROM.

Parameters

<i>buffer</i>	pointer to a 256 byte buffer that will contain the formatted OEM data fields read from the device's internal descriptors. This is raw binary data.
<i>size</i>	pointer to an unsigned long integer that contains size of the <i>buffer</i> in bytes. Upon successful completion, this will contain the number of bytes returned in the buffer.

Comments

If necessary, this function will install the INF file specified, copying the needed driver files to the Windows System directory and initiate a driver swap causing the Windows or OEM specific .sys driver to be replaced by the SMSC DFU driver. The function then uses the SMSC DFU driver to send vendor specific commands to the device firmware to read it's internal descriptors. The data is copied to the specified buffer.

The function returns zero for success, or one of the error codes described in the dfuDLL.h header file.

API Functions

Int32 End Descriptor Read (char* originalDriverInfName)

This function terminates the process of reading the device's descriptors and restores the original application driver.

Parameters

originalDriverInfName pointer to a NULL terminated string that describes the file name only (not path) of the INF file used to enumerate the device in its original application state (i.e, "usbstor.inf")

Comments

Call this function when finished reading descriptors of all devices. This function swaps the DFU driver out of the operating system and restores the original application device driver. You can plug in other devices to be read BEFORE calling this function last. This function serves as the terminating call to reading descriptors of all devices. Before calling this function, the DFU is fully installed and used for each device plugged in.

The function returns zero for success, or one of the error codes described in the dfuDLL.h header file.

Using the USB97C210 Custom Icons Package

The USB97C210 custom icons package allows OEMs to assign custom icons to the drives associated with the USB97C210 flash media controller. This allows the end user to easily distinguish between the different media types in Windows Explorer. The application works with Windows 98 SE, Windows Me, Windows 2000 and Windows XP (SP1).

Contents of the USB97C210 Custom Icons Package

The USB97C210 Custom Icons Package consists of the following:

SetIcon.exe- The custom icon application.

Oem_0424.ini- A sample Windows 98 ini file.

Smsc.ini- A sample Windows Me/2000 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 Write210.exe.

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 98 SE - The name of the ini file should be of the type "Oem_xxxx.ini" where xxxx is the VID as a hexadecimal number.

Example: If VID is 0x0424, the ini filename should be "Oem_0424.ini"

Windows Me, 2000 and XP (SP1)- 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 Write210 utility 'Option 1'. See the "Programming the Serial EEPROM" section of this document for more details.)

(**Note:** For Windows Me alone, all blank spaces (" ") in the Manufacturer's string should be replaced with under scores ("_") in the ini file name.)

Example: If MFG string is "S M S C", the ini filename for Windows Me should be "S_M_S_C.ini" and for Windows 2000, it should be "S M S C.ini"

Creating the Required SetIcon Ini Files (Cont.)**2) Setting the Ini Section Name:**

Windows 98 SE - The name of the section should be of the type [xxxx] where xxxx is the PID as hexadecimal number.

Example: If PID is 0x20FC, the ini section name should be [20FC]

Windows Me, 2000 and XP (SP1)- 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 "210 USB Controller", the section name should be "[210 U]"

Example: If the Product ID string is "210US", the section name should be "[210US]"

Example: If the Product ID string is "210", the section name should be "[210]"

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

(**Note:** The Manufacturer's string may be set or viewed using the Write210 utility 'Option 1'. See the "Programming the Serial EEPROM" section of this document for more details.)

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

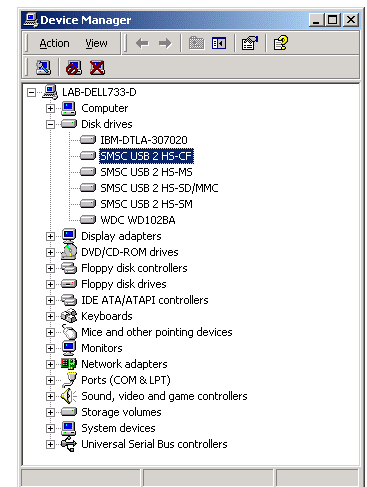
Example: CFLABEL=Compact Flash Drive

Example: SD/MMC= C:\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

**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 98 SE - "Windows\System"

Windows Me - "Windows\System"

Windows 2000 - "Windows\System32"

Windows XP (SP1) - "Windows\System32"

Manually Installing the Custom Icons Application Files

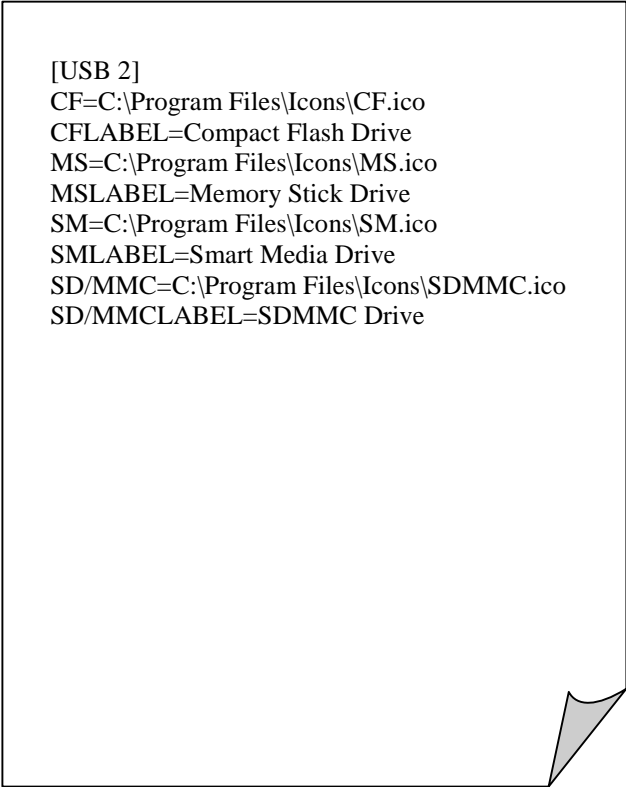
In order to perform a manual installation of the custom icons application files, the following steps should be performed:

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



```
[USB 2]
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
```

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 USB97C210 Custom Icons Application Package, a sample Windows installer "Icons.exe" is included which demonstrates a practical example of using a Windows installer to install, setup and run the Custom Icons application. To use the "Icons.exe" 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 210 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 use the Write210 utility to program the eeprom to match SMSC's VID/PID, Manufacturers String, and Product ID String for the 210. 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 known bug in Windows XP where the registry entries for the drive icons are not refreshed on a host reboot. To workaround this issue detach and reattach the USB cable, or either insert or eject the flash media. Either event will cause XP to update the icons.

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) only. 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.

Features:

1. Firmware update.
2. Descriptor (256 byte EEPROM) update.
3. Read descriptor (256 byte EEPROM) 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)

Creating the PLDU ini File

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

DFUVID = VID

This is the VID (Vendor ID) of the device whose descriptor / firmware is to be updated. The VID is specified as a four digit hexadecimal number.

DFUPID = PID

This is the PID (Product ID) of the device whose descriptor / firmware is to be updated. The PID is specified as a four digit hexadecimal number.

DFUDID = DID

This is the DID (Device ID) of the device whose descriptor / firmware is to be updated. The DID is specified as a four digit hexadecimal number.

INF = path to Smscdfu.inf

Specifies the full path to the ‘Smscdfu.inf’ file that is to be used during swapping of Mass storage class driver to the DFU driver. This inf file must contain a VID and PID entry for the device you are programming, or the DFU driver swap will not execute and the operation will fail. An example entry for a device with a VID of “0424” and a PID of “20FC” is:

```
%Smsc.DFU.Desc% = SMSCDFU.Install, USB\VID_0424&PID_20fc
```

You should change the values above in **bold** to match the VID and PID of your device.

SYS = path to Smscdfu.sys

Specifies the full path to the ‘Smscdfu.sys’ DFU driver.

DLL = path to Drvlib.dll

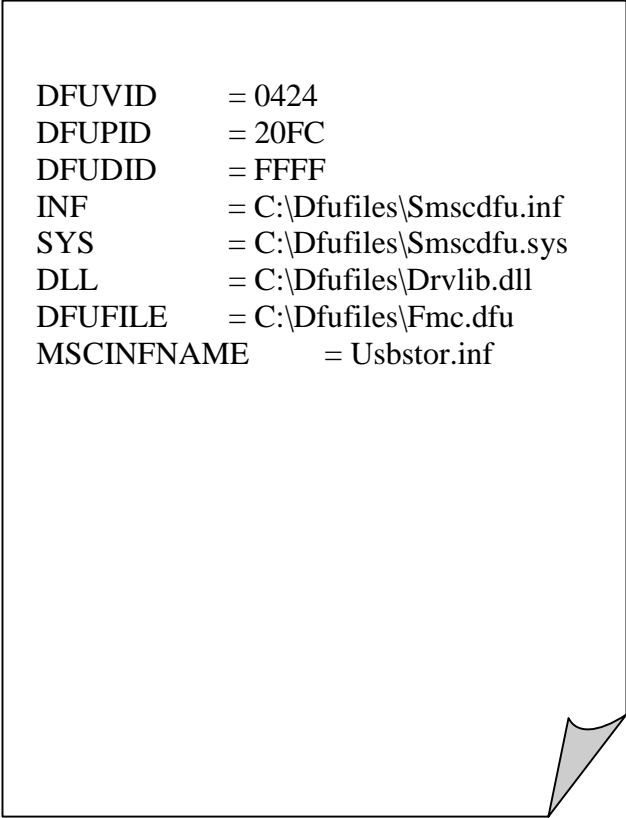
Specifies the full path to the ‘Drvlib.dll’ file that is to be used during swapping of the mass storage class driver to the SMSC DFU driver.

DFUFILE = Path to DFU file

Specifies the full path to the DFU file that is used for firmware update.

MSCINFNAME = Mass storage class Inf name

Specifies the name of the original Mass storage class driver’s INF file name. This is used while swapping the DFU driver back to the original MSC driver.

A Sample PLDU ini File

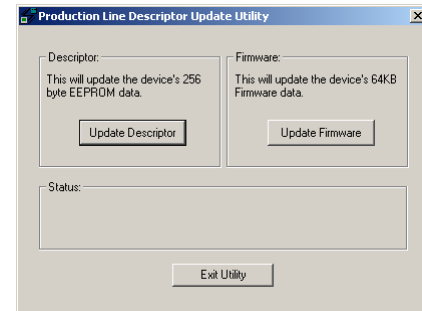
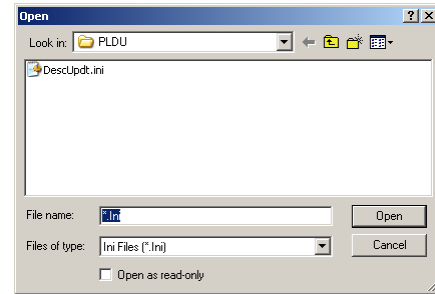
```
DFUVID      = 0424
DFUPID      = 20FC
DFUDID      = FFFF
INF          = C:\Dfufiles\Smscdfu.inf
SYS          = C:\Dfufiles\Smscdfu.sys
DLL          = C:\Dfufiles\Drvlib.dll
DFUFILE      = C:\Dfufiles\Fmc.dfu
MSCINFNAME   = Usbstor.inf
```

NOTE:

- i. There can be spaces before and after the '=' (equals) sign, but the total number of characters per line (including spaces) should be LESS THAN (<) 255.
- ii. All the paths specified above should be valid, as the application will make sure that those files do exist in their respective paths. If a path is not valid, then the application would display a corresponding ERROR message and terminate itself.

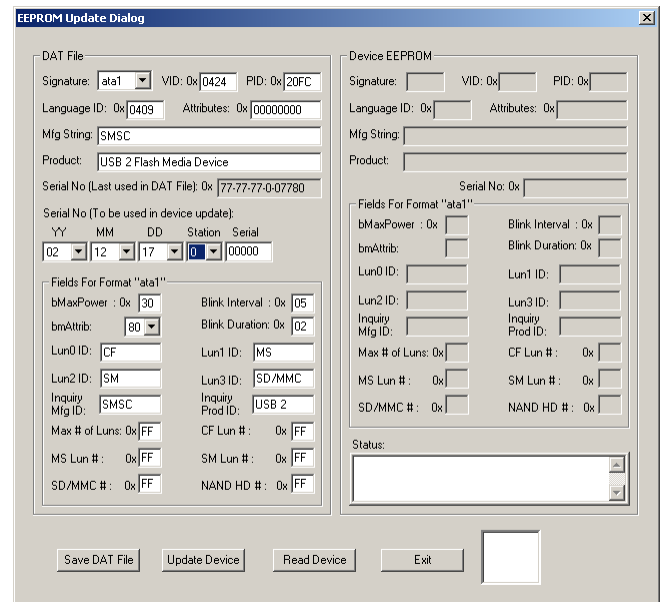
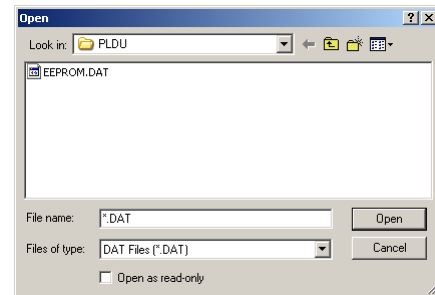
Setting Up the PLDU Application

1. First attach a USB97C210 device to the host. To start the PLDU application, simply double click “DescUpdt.exe” executable. The application will prompt you to select the location of the ini file.
2. Provided the ini file contains the correct path to the key files on the local machine, the main program dialog opens. Here you are given two options:
 - a. Update Descriptors- Updates NVStore data such a VID/PID, Manufacturer and Product ID strings from the “EEPROM.DAT” file.
 - b. Update Firmware- Updates the device firmware using a DFU update file with the .dfu extension.



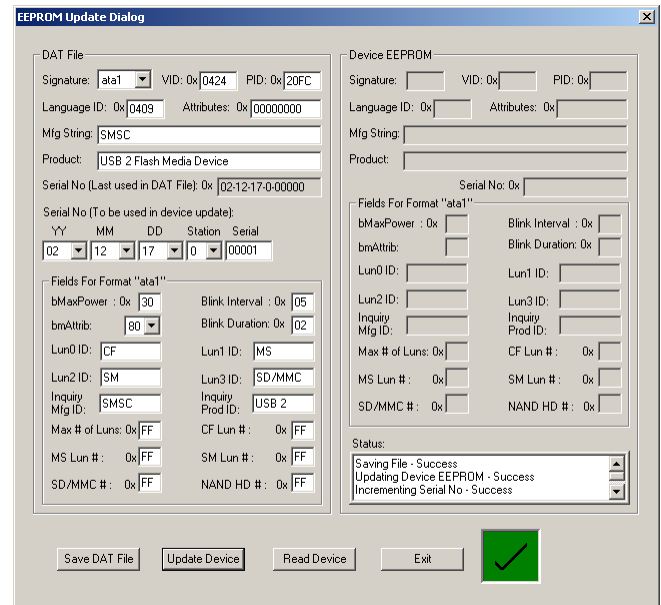
Using the PLDU to Update Device Descriptors

1. The first operation that should be performed on a USB97C210 device coming off the production line is to update its descriptors. To do this, press the “Update Descriptors” button on the main dialog above. For the first device only, 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 program will swap the mass storage class driver for the SMSC DFU driver.
2. Once the DFU driver swap has completed, the programming dialog appears. At this point the station is setup and ready to begin programming USB97C210 devices.

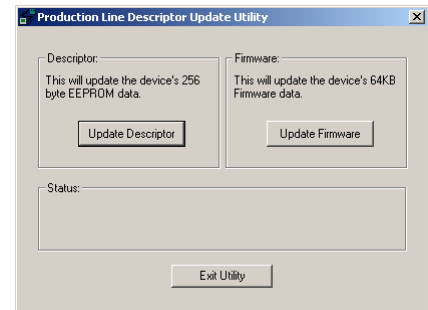


Using the PLDU to Update Device Descriptors (Cont.)

- To program the first device, the operator simply presses the “Update Device” button. Once the Update Device button is pressed, the application saves all of the data in the editable fields (the fields with a white background) including the serial number, to the EEPROM.DAT file. After that, all of the 256 bytes of data contained in the EEPROM.DAT file is programmed into the device. The operation takes about 12 seconds to complete. Provided the programming was successful, the EEPROM Update Dialog displays a green box with a black 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.

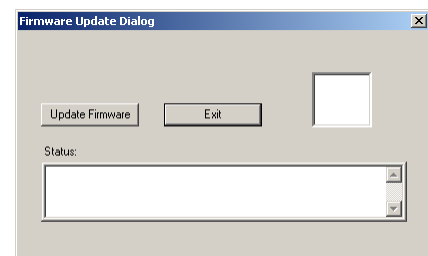


- Once all devices have been programmed, the user selects the “Exit” button to return to the main dialog.



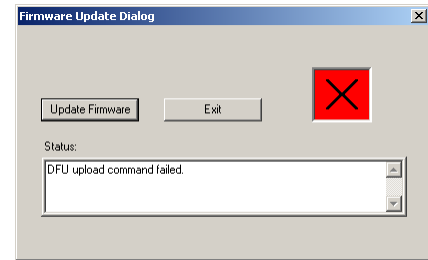
Using the PLDU to Update Firmware

- The PLDU application can also be used on the production line to apply in circuit firmware updates through the USB. In order to do this you **MUST** have flash ROMs preprogrammed with the DFU loader. (See the instructions on creating the “BOTH.BIN” file in the USB97C210/211 DFU Release Notes.)
- Assuming you have completed steps 1 and 2 in the “Setting Up the PLDU Application” section, to initiate a firmware update, the user will press the “Update Firmware” button. The dialog on the right appears. At this point the station is ready to begin updating device firmware.

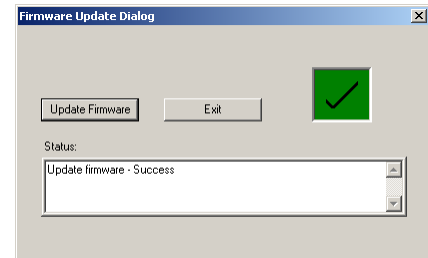


Using the PLDU to Update Firmware (Cont.)

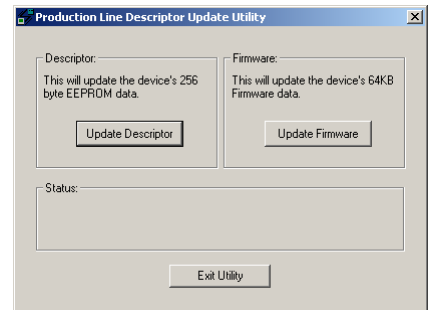
3. To initialize a firmware update, the user presses the “Update Firmware” button on the Firmware Update Dialog. If the application displays a red box with an “X” in it as shown on the right, the operation failed. If this is the case, be sure to check that you have the DFU loader programmed into the chip. (See the section on creating the “BOTH.BIN” file in the USB97C210/211 DFU Release Notes.



4. If the operation completes successfully, the Firmware Update Dialog displays a green box with a black checkmark to indicate success. The user may now remove the device, attach the next device and repeat the programming operation.



5. Once all devices have been updated, the user presses the “Exit” button to return to the main dialog.



Using the Production Line Test Utility (PLTU)

Purpose: The PLTU application is used to test the basic functionality of USB97C210 devices in a production line environment using Windows 2000 (SP3) only. 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      = 20FC
INQUIRY_MFG  = SMSC
INQUIRY_PRODUCT = 210
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
```

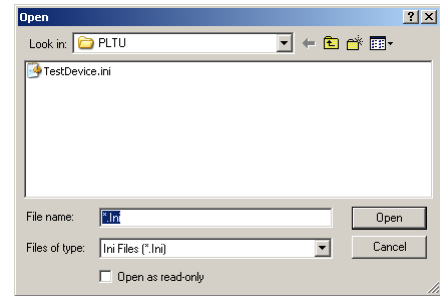
```
DEV5_LUN0 =
DEV5_LUN1 =
DEV5_LUN2 =
DEV5_LUN3 =
```

NOTE:

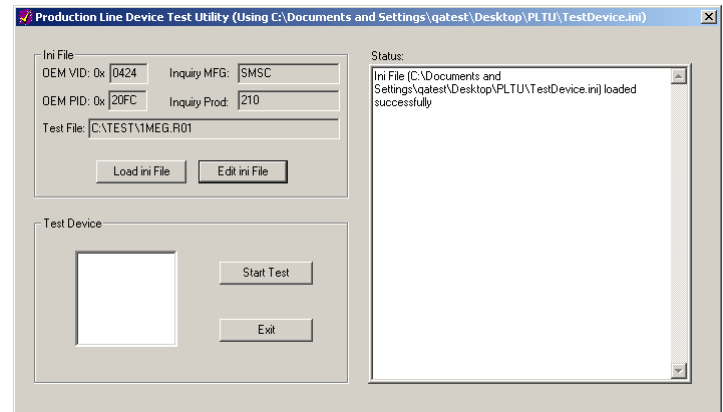
There can be spaces before and after the '=' sign, but the total number of characters for an entire line (including spaces) should be less than 255.

Setting Up the PLTU Application

1. First attach a USB97C210 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.

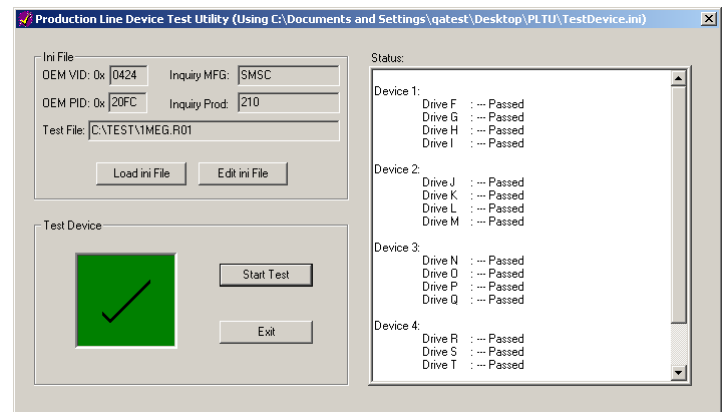


2. Provided the ini file contains the correct path to the key files on the local machine, the main program dialog opens. The station is now ready to begin testing devices. At this point you should attach the devices to be tested and ensure that they have good media with sufficient free space to hold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



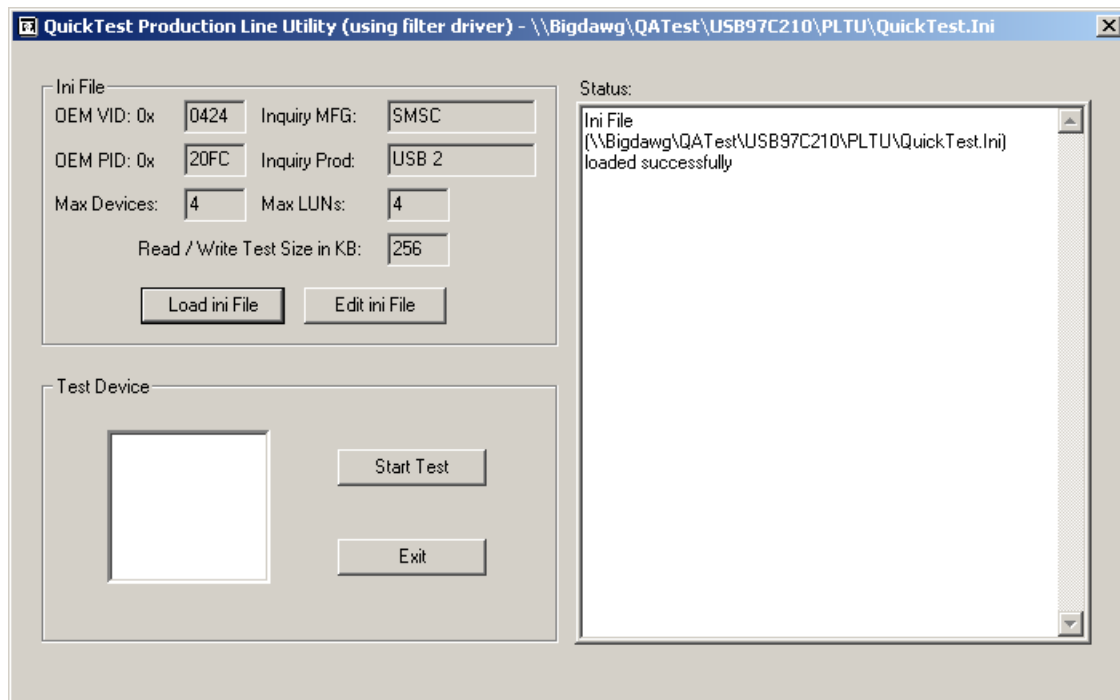
2. Once the test has completed, the user should remove all of the tested devices and then attach the next set of devices to be tested. Once all of the devices are attached and enumerated (as indicated by the presence of drive icons in Windows Explorer), the user repeats step 1 to test the next set of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

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) USB97C210 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.



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.

Known Issues with the USB97C210 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
In order to perform a firmware update, the flash rom must be preprogrammed with the DFU loader code.	See the USB97C210/211 DFU Release Notes section on creating the "BOTH.BIN" file.	N/A
In the main dialog window of the PLDU application, when the "Update Descriptor" or "Update Firmware" is clicked, the application swaps the Mass storage class driver with the DFU driver before opening the corresponding dialog box. This requires that the device be connected before the user can click on these buttons. If no device is connected, the driver swap and consequently the update operation will fail.	Make sure that there is a device connected BEFORE attempting to perform either a Descriptor or Firmware Update.	N/A
In the PLDU application, When the user exits either from the "EEPROM Update Dialog" or the "Firmware Update Dialog", the application tries to restore the Mass storage class driver before exiting the dialog. This requires that the device be connected while the application exits these dialog boxes. If the device is removed before the application exits, the application will prompt for the user to reconnect the device.	Make sure the device is connected to the host before exiting the EEPROM or Firmware Update dialog screens. A device must be connected while the DFU driver swap takes place for the operation to complete successfully.	N/A

Using the EPRMUPDT.exe Utility

EPRMUPDT.exe is a DOS based utility used to write and / or read EEPROM data to / from the USB97C210 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"]
-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
-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 and -r options is optional. If the path names do 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 and -r options are to be specified with full path information. If the files are in the current directory, then the path information is not necessary.

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.
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, connect a new device, and enter the serial number again.

Using the EPRMUPDT.exe Utility (cont.)

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

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

Limitations of the EPRMUPDT.exe Utility:

1. Only supports devices connected to a UHCI host controller.
2. The UHCI host controller to which the device is connected should be the first one in the enumeration order.
3. Only supports devices connected at the root hub level.
4. The first MSC device in the enumeration order has to be the device whose EEPROM data is to be updated or read.

Using the CheckROM.exe Utility

CheckROM.exe is a DOS based utility used to check the NVStore data of USB97C210 device against a user specified template DAT file. This utility also checks the device's firmware version against a specified version number. This utility is designed to be used by OEMs to streamline their production environment.

CheckROM Usage:

```
CheckROM [-h|-u] [-v] [-e"DATFileName"] [-f"version"]
-h|-u          print help/usage
-v            verbose, optional, default is off
-e"DATFileName" name of DAT file (with full path) that is to be checked against the device EEPROM
-f"version"    version number that is to be checked against the firmware version of the device
-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 name for -e option 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 name for the -e option is 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.

Features:

1. Reads the contents of the device EEPROM and checks the entire contents (excluding serial number) against the specified template DAT file.
2. Reads the firmware version of the device and checks that against the specified version number.
3. The options for checking EEPROM data or firmware version number can be specified together or alone.
4. Provides an option (-v) to turn on or off the additional debug / status comments.
5. Allows checking 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.
6. 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 checking the EEPROM or firmware version of 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 checking the EEPROM or firmware version of the device. This can happen if no matching devices are found, or error while reading EEPROM data from device, or the EEPROM check or firmware version check 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 exit code is ignored and the application proceeds to prompt for checking the next device.

"PASS" - Means no error occurred and the process of checking EEPROM and / or firmware version of the device completed successfully and the application exits with a return code of 0. If the -i option is specified, then the application proceeds to prompt for checking the next device. At this stage, the user can safely remove the existing device and connect a new device for checking.

Using the CheckRom.exe Utility (cont.)

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

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

Limitations:

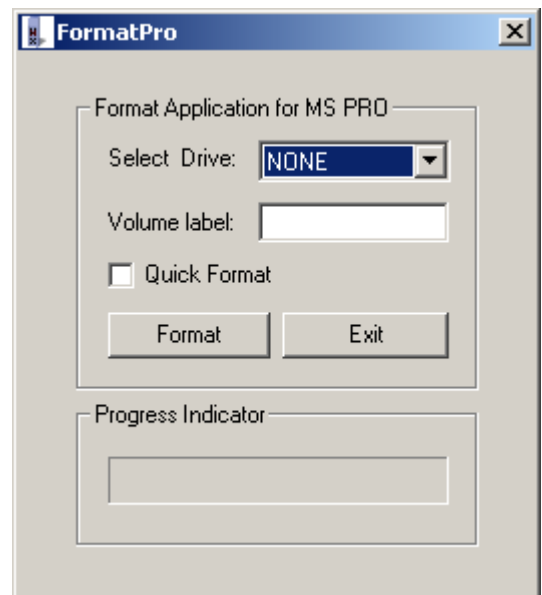
1. Supports only devices connected to a UHCI host controller.
2. The UHCI host controller to which the device is connected should be the first one in the enumeration order.
3. Supports only devices connected at the root hub level.
4. The first MSC device in the enumeration order has to be the device whose NVStore data is to be checked.

Using the MSPro Format Utility

SMSC provides a utility "FormatPro.exe" to apply the special Sony proprietary format to MSPro media. Although MSPro cards can be formatted using the normal Windows format, using the Sony format will result in significantly higher read/write performance.

To use the application, make sure the USB97C223 device is enumerated and there is MSPro media inserted in the card slot. Start the application, select the drive from the drop-down menu and specify a volume label if desired. Check or uncheck the quick format box, and then press the format button to begin the format. The application will notify the user once it has completed the operation.

Important Note: In order to use the FormatPro application in either Windows 98 or Me, the SMSC Password Filter driver must be loaded. To do this, go to the device manager and double click the device entry for the 223 in the USB section. Select "Update Driver" and then follow the update driver wizard. When asked for the location of the new driver, point to the folder containing the SMSC Password Filter driver and inf file.



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____210_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 = "210 USB97C210" (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 Windows 2000 or 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 devices under Windows 2000 (SP2 and earlier) to disassociate the device from the OEM MSC driver. This allows the SMSC Win2K driver to be loaded. 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.

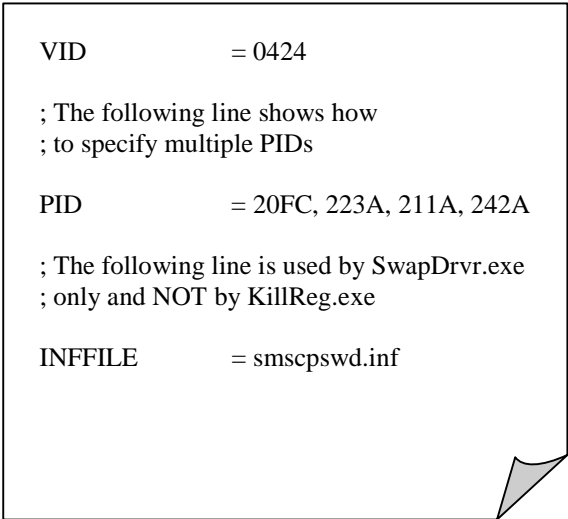
INI File Requirements:

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, 242A  
  
; 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. Unfortunately, SwapDrv does not work with Windows 98 and Me. The only USB97C210 application that requires the password filter driver be loaded when running XP is the QuickTest production line test utility. If you are not using that utility 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.

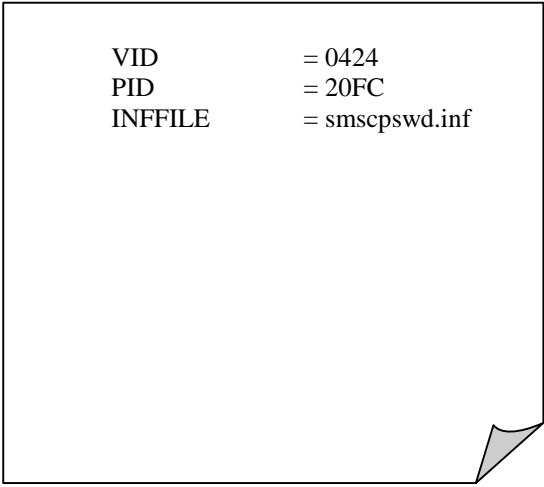
INI File Requirements:

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          = 20FC  
INFFILE      = smscpswd.inf
```

Using the USB97C210 with Linux

Versions 2.4.20 and greater of the Linux kernel provide native support for multi-LUN USB mass storage class devices like the USB97C210. Some brands of Linux such as SuSe 8.2 require little or no user setup. Simply plug in your USB97C210 device, and icons will appear, provided there is media in the card reader slots. Other brands of Linux such as Redhat require the user to configure the kernel in order to enable multi-LUN support in the mass storage class driver. The procedure for doing that is:

Requirement:

RedHat Linux 9.0 with kernel 2.4.20 or greater

Steps:

1. Install RedHat Linux 9.0 on the host system
2. Login to the system as 'root'.
3. Open a terminal window.
4. Plug the multi-LUN card reader into the host.
5. At the shell prompt, type 'cat /proc/scsi/scsi'.
6. If the screen shows only one LUN, type 'lsmod'.
7. If 'usb-storage' does not exist, type 'insmod usb-storage'.
8. If 'usb-storage' exists, type 'cdrecord -scanbus'. It will display
scsibus0:

```

0,0,00) 'SMSC          ' '210 U HS-CF' 'X.XX' Removable Disk
0,1,01) *
0,2,02) *
0,3,03) *
0,4,04) *
0,5,05) *
0,6,06) *
0,7,07) *
```
9. Create a batch file with the following calls:

```

'echo "set-single-device 0 0 0 0">/proc/scsi/scsi
'echo "set-single-device 0 0 0 1">/proc/scsi/scsi
'echo "set-single-device 0 0 0 2">/proc/scsi/scsi
'echo "set-single-device 0 0 0 3">/proc/scsi/scsi
'cat /proc/scsi/scsi'
```
10. After running the batch file, the screen should display:

Attached devices:

Host: scsi0 Channel: 00 ID: 00 LUN: 00

Vendor: SMSC Model: 210 U HS-CF

Type: Direct-Access

Rev: X.XX

ANSI SCSI revision: 02

Attached devices:

Host: scsi0 Channel: 00 ID: 00 LUN: 01

Vendor: SMSC Model: 210 U HS-MS

Type: Direct-Access

Rev: X.XX

ANSI SCSI revision: 02

Attached devices:

Host: scsi0 Channel: 00 ID: 00 LUN: 02

Vendor: SMSC Model: 210 U HS-SM

Type: Direct-Access

Rev: X.XX

ANSI SCSI revision: 02

Attached devices:

Host: scsi0 Channel: 00 ID: 00 LUN: 03

Vendor: SMSC Model: 210 U HS-SD/MMC

Type: Direct-Access

Rev: X.XX

ANSI SCSI revision: 02

11. Now multi-LUN support is enabled and you should be able to mount and access all media normally.

Media Tested with the USB97C210

The following flash media cards were used during the development and testing of the USB97C210. All media listed has been determined to work properly and be compatible with the USB97C210.

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

USB97C210 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.

Media Used for Testing:

Lexar Media (24x) 256MB

Lexar Media 128MB

Sony 1GB

Memorex 128MB

Buffalo 256MB

Lexar Media 64MB

Typical USB 1.1 device benchmark data			
Media	Read KB/sec		Write KB/sec
CF	1,000 KB/s		675 KB/s
SM	806 KB/s		384 KB/s
SD	925 KB/s		625 KB/s
MMC	886 KB/s		360 KB/s
MS	694 KB/s		330 KB/s

MEASURED (Sustained Data rates Bytes/sec) Full Speed--4MB file size

Full Speed (USB1.1)				
Media	FS Mult.	Read KB/sec	FS Mult.	Write KB/sec
CF	1.0	1,030 KB/s	1.3	890 KB/s
SM	1.2	973 KB/s	1.3	488 KB/s
SD	0.2	194 KB/s	0.4	246 KB/s
MMC	0.2	176 KB/s	0.6	219 KB/s
MS	0.9	632 KB/s	1.4	471 KB/s
MS Pro	N/A	600 KB/s	N/A	421 KB/s

FS MULT is ratio to 1.1 reader device measured speeds

MEASURED (Sustained Data rates Bytes/sec) High Speed--4MB file size

High Speed (USB2.0)				
Media	FS Mult.	Read KB/sec	FS Mult.	Write KB/sec
CF	5.4	5,446 KB/s	4.4	2,985 KB/s
SM	6.2	5,000 KB/s	1.9	742 KB/s
SD	7.6	7,070 KB/s	4.1	2,590 KB/s
MMC	1.7	1,510 KB/s	1.3	471 KB/s
MS	1.2	858 KB/s	2.0	659 KB/s
MS Pro	N/A	864 KB/s	N/A	547 KB/s

FS MULT is ratio to 1.1 reader device measured speeds

GPIO Assignment Table

The following is a table of GPIO assignments as configured and compiled for this version of the firmware.

USB97C210		
Name	Description	Notes
GPIO0	Flash Media Activity LED	Indicates media activity. GPIO high indicates activity.
GPIO1	Unused	
GPIO2	EE_CS	EE Chip select
GPIO3	V_BUS	USB V Bus detect
GPIO4	EE_DIN/EE_DOUT	EE Data input/output
GPIO5	HS Ind./SD Card Detect	HS ind. or SD Crd. detect, GPIO high indicates HS+blink active
GPIO6	A16	A16 address line connect for DFU or debug LED
GPIO7	EE_CLK	EE Clock input
GPIO8	MS Power Control	Controls transistor to switch power to the memory stick
GPIO9	CF Power Control	Controls transistor to switch power to the compact flash
GPIO10	SM Power Control	Controls transistor to switch power to smart media
GPIO11	SD Power Control	Controls transistor to switch power to SD/MMC
GPIO12	MS Activity	Indicates media activity. GPIO high indicates activity.
GPIO13	CF Activity	Indicates media activity. GPIO high indicates activity.
GPIO14	SM Activity	Indicates media activity. GPIO high indicates activity.
GPIO15	SD/MMC Activity	Indicates media activity. GPIO high indicates activity.

Known USB97C210 Firmware Related Issues**General:**

Issue:	Workaround:	Status:
None.	None.	

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 USB97C210 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 USB97C210, 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 USB97C210 device may fail to recognize an SD/MMC card inserted while writing to either CF or MS or SM cards.	Reinsert the card after the transfer has completed.	Use workaround.

xD Devices:

Issue:	Workaround:	Status:
No known issues.		

Issues Not Related to USB97C210 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 USB97C210 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 "Write210.exe" utility.	
Surprise removal of the USB cable during a write to any media type under Windows 98 or Me, sometimes causes the host to become unresponsive.	Reboot the host.	This appears to be a bug with the operating systems. All mass storage class devices tested have displayed this same behavior.
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. The USB97C210 firmware continues to function normally however. This can be verified by reenumerating it on a 2 nd computer
Windows 2000 does not immediately report that media is write protected when attempting to perform a full format. The format will appear to progress to completion, but at the end of the operation reports that the media is write protected.	None.	This is normal behavior for Windows 2000. This occurs for all USB write protectable devices when attempting to perform a full format.
16MB MMC media reports an incorrect format capacity when you attempt to format it in Windows 98 or Me after having previously formatted a 64MB MMC.	Power cycle the board.	This appears to be a bug with the Windows Operating system.
Prematurely attempting to access a drive after resuming from suspend sometimes results in a device I/O error in Win2K. This is a known issue at Microsoft. (Reference Microsoft Knowledge Base article Q323754)	Obtain and install the updated Usbhub.sys file from the hotfix that is described in Microsoft Knowledge Base article Q306455.	N/A
Under Mac OS 9.x only one drive will appear on the desktop. This is normal as the Mac OS 9.x mass storage class driver does not support multiple LUN devices.	Use the MacOS 8.6-9.x driver provided by SMSC.	Use the MacOS 8.6-9.x driver provided by SMSC.
The MacOS MSC driver does not handle surprise removal of the USB cable during writes properly.	None.	Currently under investigation.
MacOS 9 reports an error when the 210 device is resumed from a suspend state, informing the user that the media was removed.	Close the error message and the 210 will operate normally.	Currently under investigation.