DLPC\_API\_LIB 654X Sample Code Documentation

Beta 1 - 12/2020

Contents

[Purpose: 2](#_Toc58846449)

[Overview: 2](#_Toc58846450)

[Building/running: 2](#_Toc58846451)

[IO: 2](#_Toc58846452)

[Command packet: 3](#_Toc58846453)

[Reading data: 4](#_Toc58846454)

[Response packet: 5](#_Toc58846455)

[Used Commands: 6](#_Toc58846456)

[Flash procedure: 6](#_Toc58846457)

# Purpose:

This sample code is intended to serve as a reference for how to update the flash sectors of a 654x-based or 754x-based projector system.

# Overview:

This sample code leverages the dlpc\_common.h and dlpc\_common\_private.h header for the majority of command operations. Use of these headers requires a call to DLPC\_COMMON\_InitCommandLibrary(…) to provide read/write buffers as well as defining read/write callbacks.

dlpc654x\_sample.c provides a sample implementation of the business logic required to update the flash image on the projector system.

win\_io.c provides an abstraction layer for interacting with the WinUSB interface. The ioInit() function must be called in order to initialize WinUSB settings and the connection to the 654x device. win\_io.h will need to be updated with the correct hardware identifiers to connect to other devices.

Flash programming may be performed while the projector system is in bootloader mode. All flash sectors may be written at once, or you can update specific sectors only. You may additionally choose whether to update the bootloader or not.

# Building/running:

This sample code is provided as a CMake project, with a full implementation provided for Windows/Visual Studio. The project is also able to be built on Linux, which relies on having libusb-1.0.0-dev installed with libusb.h in /usr/include/libusb-1.0, or a standard header install location.

On Windows, assuming you’ve configured your Visual Studio installation to support CMake (see here: <https://docs.microsoft.com/en-us/cpp/build/cmake-projects-in-visual-studio?view=msvc-160>), you can simply open the project directory with Visual Studio and it will build it for you.

On Linux, you can follow the standard CMake build procedure:

1. Navigate into the source directory: cd /path/to/dlpc654x\_samples
2. Create and navigate into a build directory: mkdir build && cd build
3. Run CMake to generate the Makefiles: cmake ../ -DPLATFORM=64
4. Build the project: make
5. Run the executable: ./dlpc654x\_samples /path/to/image/file [options]
   1. Options include:
      1. -m: update **M**odified sectors only
      2. -b: Program **B**ootloader (skipped by default)

To run the program, you simply invoke the executable and provide a path to a flash image file as the first argument, followed by desired options. For example, ./dlpc654x\_sample.exe C:/path/to/FlashImg.img -m

The 654x projector should be identified and connected for you.

# IO:

This sample code provides win\_io.c and win\_io.h to support WinUSB connections on Windows. unix\_io.c and unix\_io.h are provided to support LibUSB connections on Linux.

When sending read and write commands, we need to prefix each command with a header. The full command, including header, is formatted as below:

# Command packet:



For the purposes of this sample code, we set the header fields as follows:

|  |  |
| --- | --- |
| **Field** | **Value** |
| Destination | 1 |
| OpcodeLen | 0 |
| DataLenP | 1 |
| ChksumP | 0 |
| ReplyReq | 1 |
| Read | When writing: 0  When reading: 1 |

Because of this, when performing a write command, the command is formatted as:

0x51 <1-byte opcode> <2-byte data length> <data>

And when reading, the data is formatted as:

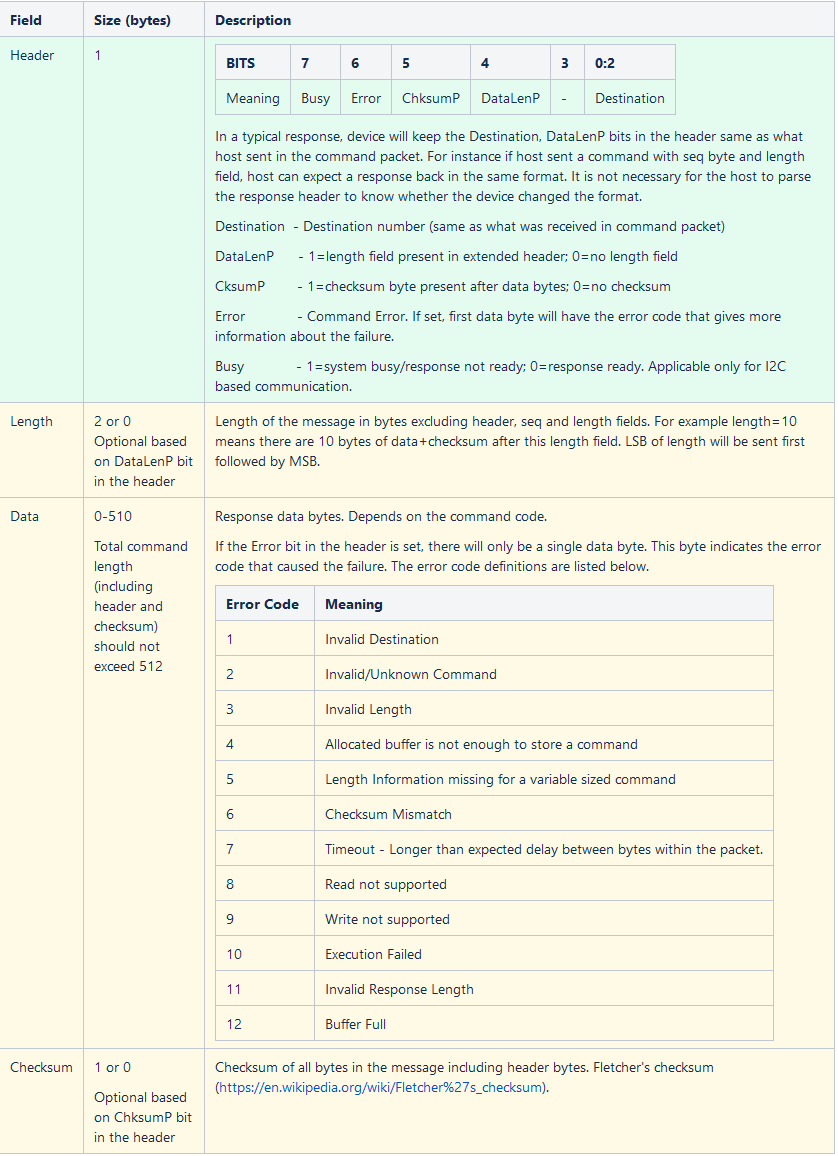
0xD1 <1-byte opcode> <2-byte data length> <data>

# Reading data:

This system supports both fixed-length and variable-length reads. For a fixed-length read you can simply read the full, expected length (including header length and 2 bytes for the length field, if DataLenP was set in write command header) at once, or you can split it into one read of header length and a second read of the data length.

Variable length reads must be read in two steps. This is because the data length cannot be known without checking the data length field from the response header. After reading the header and 2 bytes for the length field (3 bytes total), you can proceed to read the rest of the data from the response packet (see below).

# Response packet:



# Used Commands:

The following commands are used in the flash update process.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Opcode** | **Arguments** | **Summary** |
| ReadMode | 0x00 | N/a | Reads the current mode  0x00 = booatloader mode  0x01 = main application |
| WriteSwitchMode | 0x02 | 1-byte mode  0x00 = to boot mode  0x01 = to app via reset  0x02 = to app direct | Commands system to change modes |
| ReadGetFlashSectorInformation | 0x21 | N/a | Gets information about the system’s flash storage (sector start addresses and size)  Number of distinct flash sector types = floor(number of bytes read / 6)  Sector size = first 4 bytes returned  Number of sectors = next 2 bytes returned |
| ReadChecksum | 0x26 | <4-byte start address> <4-byte number of bytes> | Calculates and reads the checksum for the given address+X bytes |
| WriteUnlockFlashForUpdate | 0x22 | 4-byte command  0x00000000 = lock  0xF7A54027 = unlock | Unlocks/Locks the system for flash updating |
| WriteEraseSector | 0x23 | <4-byte sector address> | Erases the sector at the given start address |
| WriteInitializeFlashReadWriteSettings | 0x24 | <4-byte start address> <4-byte number of bytes> | Marks the bytes at address+X bytes for writing (see WriteFlashWrite) |
| WriteFlashWrite | 0x25 | <data> | Writes the given data to the location marked by WriteInitializeFlashReadWriteSettings |

# Flash procedure:

1. Read current mode (ReadMode command). While not in boot mode:
   1. Send WriteSwitchMode command
2. Gather flash sector information (GetFlashSectorInformation command)
3. For each sector, seek to that address in the flash image file. Find and store the address that contains the flash table signature (0xF7A547AB7E5162A7)
4. If flashing \*only modified\* sectors:
   1. Calculate checksum for every sector in the flash image file
   2. Read checksum for every sector in the system flash (ReadChecksum command)
   3. If the checksums differ, add the sector to flash list
5. If flashing \*all\* sectors:
   1. Add all sectors to flash list
6. Unlock the system flash for updating (UnlockFlashForUpdate command)
7. For every sector in flash list:
   1. Erase that sector on the system flash (EraseSector command)
   2. Configure flash settings, telling the system which flash sector and length that’s about to be updated (InitializeFlashReadWriteSettings command)
   3. Write new flash data (FlashWrite command)
8. Write the flash table:
   1. Configure flash settings, telling the system which flash sector and length that’s about to be updated (InitializeFlashReadWriteSettings command)
   2. Write new flash data (FlashWrite command)
9. Lock the system flash (UnlockFlashForUpdate command)
10. Validate updated sectors. For each sector in flash list, compare the checksum of the flash image file sector with the checksum of the system flash sector (ReadChecksum command)
11. Boot into main application (SwitchMode command)