Manufacturing Tool Client Driver Development Guide

Version History

VERSION	DATE	AUTHOR	CHANGE DESCRIPTION
1.0	Nov 1th, 2009	Li Xingyu – b02754	Original Version
1.1	Aug 5th, 2010	Tina Zhu – r65086	

Table of Contents

1 The purpose

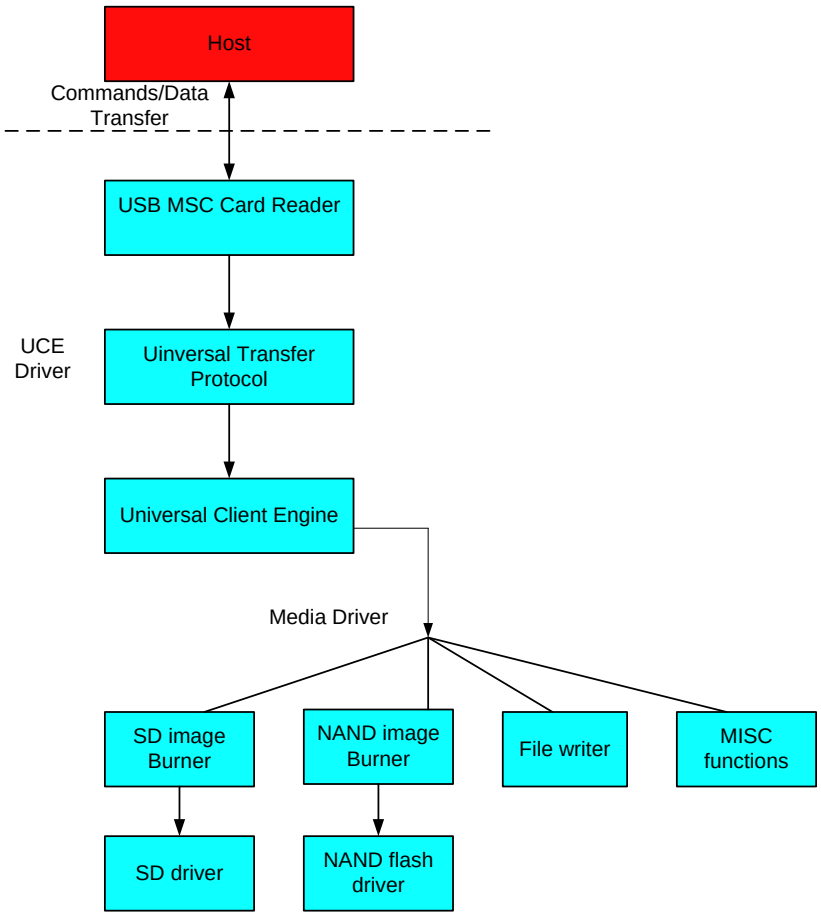
The doc gives an introduction of manufacturing tool (also called universal updater tool) client driver.

2 What is functionality of manufacturing tool client driver

To implement UTP protocol and execute commands sent from host.

3 Driver architecture

The client driver consists of UCE driver and media driver.



3.1 UCE driver

UCE driver acts as a bridge to transfer commands/data between host side and device side.

3.1.1 USB MSC card reader

The layer implements USB MSC protocol and reports USB host as a card reader. To send commands or data from host to client, host must setup a way to communicate the client. USB MSC is chosen as a protocol here since it is simple and speed efficient, and Windows OS pre-installs USB MSC driver, which means we can use the driver directly.

Card reader which is a subtype of USB MSC is chosen since it can stop the disk read/write operation from host.

This layer is implemented by two files: Bot.cpp and Block.cpp.

Bot.cpp implements bulk-only transfer routine.

Block.cpp implements USB MSC card reader functions which Contains SCSI-2 direct-access device emulation implementation. In this way, access operation can be prohibited when the USB MSC device is recognized by Windows System.

There are limited SCSI commands will be supported, which are listed in STORE_IsCommandSupported function. And these commands are executed in STORE_ExecuteCommand function.

NOTE: SCSI_UTP is a vendor defined command which implements manufacturing tool specific command. Please refer to UTP.doc for detail

The module is quite similar to the sample Microsoft provides. For the more information, see the Microsoft help document - *USB Function Mass Storage Client Driver.*

For USB Mass Storage information, please check the following specifications:

Universal Serial Bus Mass Storage Class Specification Overview

USB Mass Storage Class Bulk-Only Transport

Small Computer System Interface - 2 (SCSI-2)

3.1.2 Universal transfer protocol

UTP.cpp implements the protocol which transfers data and message to UCE layer.

3.1.3 Universal client engine

The layer is contains uce.cpp, uce_media.cpp and sdboot.cpp in common folder and bspuut.cpp in bsp code folder.

Uce.cpp implements universal client engine which parses the commands sent from host and executes them with related command functions.

For these commands information, see *Manufacturing Tool UCL user manual.doc*.

Uce_media.cpp invokes corresponding media driver functions to execute the commands.

Sdboot.cpp executes the detail operations relate to SD/MMC.

Bspuut.cpp implements the specific-platform operations.

3.2 Media driver

The task manufacturing tool is to burn all kinds of contents like image, demo files to non-volatile storage devices. Media driver is used to finish the burning tasks.

Those tasks can be divided to following parts:

1. Image burning: to burn images to specified media.
2. File writing: to write files to specified media.

3. Other features of customization for customer requirement.

Currently, only NAND flash and SD/MMC media is supported.

It is to access NAND and SD/MMC media, the two medias driver must be supported firstly. And for using either one of two medias, the corresponding media item must be selected in UUT project catalog. It is the customers' responsibility to provide their own media driver like NAND flash driver, SD/MMC/eMMC/SPI NOR, etc. The way of how to burn customers' image totally depends on the requirement of customers theirselves. Anyway, the UTP protocol implementation should be identical and shouldn't be changed.

3.2.1 NAND Media

There is the NAND driver to support access NAND media firstly. Then the client driver need implement NAND boot burner part in NAND driver.
It includes NandBootBurner.c which implements some READ/WRITE operations to NAND.

For more NAND driver information, see *FSL_WSDK_CE600_ReferenceManual.pdf.*

3.2.2 SD/MMC Media

The client driver will call SD/MMC driver to write/read SD/MMC media by specialized IOCTL directly.
For more SD/MMC driver information, see *FSL_WSDK_CE600_ReferenceManual.pdf.*

4 Code organization

Code organization follows below principles:

1. Try to share common codes for each platform.

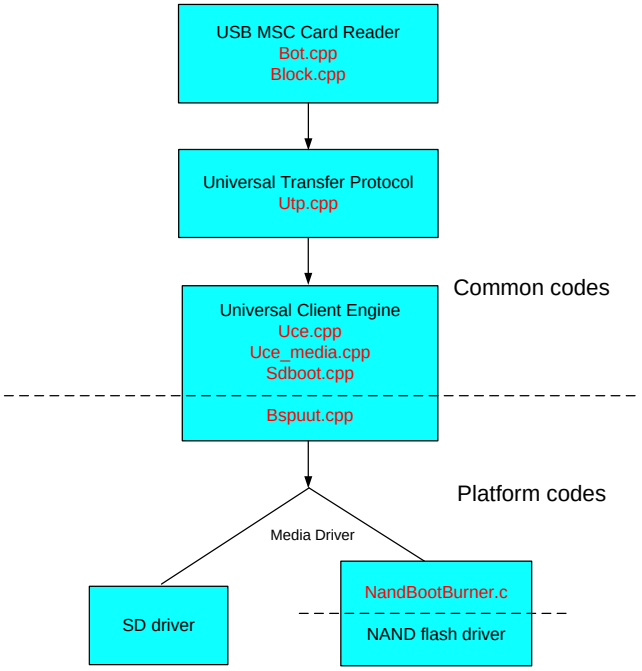
   Locates in *$(_PLATFORMROOT)\COMMON\SRC\SOC\$(_COMMONSOCDIR)\UUT.*

2. BSP codes for each platform.

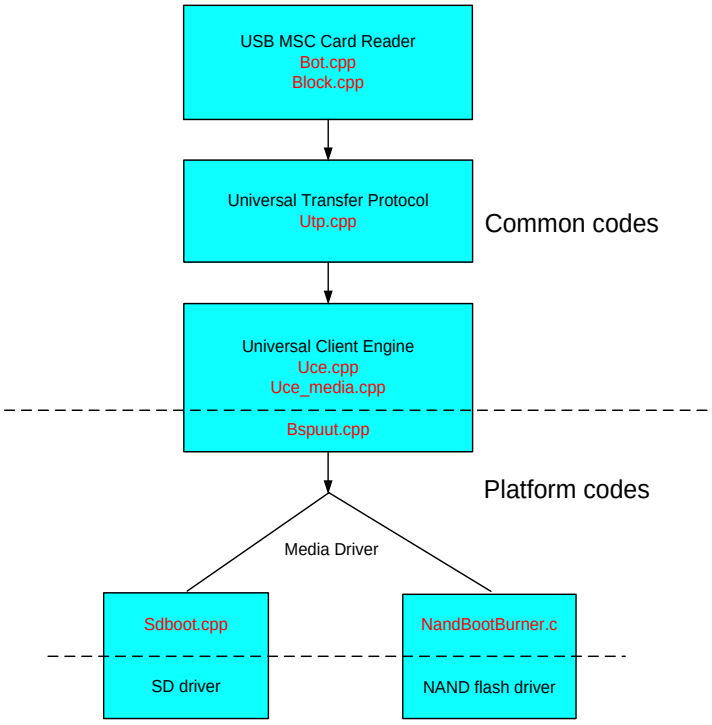
   Locates in *$(_TARGETPLATROOT)\SRC\ DRIVERS\UUT*.

3. Media driver codes for each platform.

   Only NAND Boot Burner part need be implemented in client driver. It is in the site *$(_TARGETPLATROOT)\SRC\COMMON\NANDBOOTBURNER or $(_PLATFORMROOT)\COMMON\SRC\SOC\$(_COMMONSOCDIR)\boot\fmd\nandbootburner*



NOTE: In fact, media driver is always related to platform. So these operations control media should be implemented in media driver. But the functions of interrelated SD media are completed in UCE layer now. It suggests be implemented in SD driver.



5 Client driver implementation

There are two types of i.MX devices: HID mode i.MX device and Bulk-IO mode i.MX device. Till now, i.MX233/28/508 belongs to HID mode i.MX device; i.MX35/51/53 belongs to Bulk-IO mode i.MX device.

This client driver implementation is different with the device mode. And the common codes are shared for all platforms. The main implementation is about the codes of platform and the codes of media.

5.1 Register settings

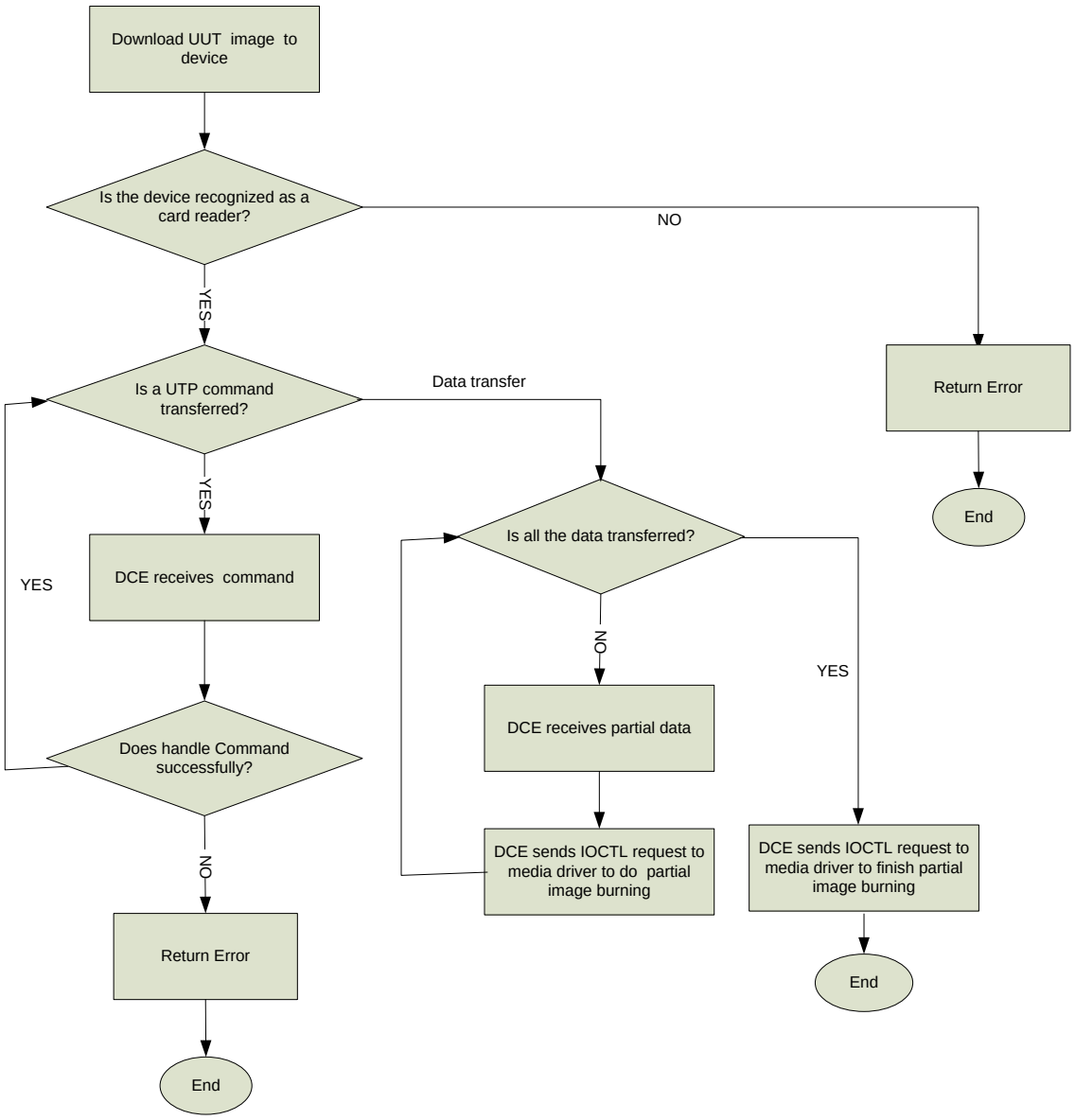
The client driver registry settings are under. The registry settings information, please refer to the Microsoft help document - *USB Function Client Driver Registry Settings*.

```
[HKEY_LOCAL_MACHINE\Drivers\USB\FunctionDrivers]
   "DefaultClientDriver"=- ; erase previous default
[HKEY_LOCAL_MACHINE\Drivers\USB\FunctionDrivers]
   "DefaultClientDriver"="UUT"

[HKEY_LOCAL_MACHINE\Drivers\USB\FunctionDrivers\UUT]
   "Dll"="UUT.dll"
   "InterfaceSubClass"=dword:06
; 06h corresponds to the USB mass storage client.
   "InterfaceProtocol"=dword:50
; 50h corresponds to bulk-only transport (BOT).
   "DeviceName"="DSK1:"
   "FriendlyName"="Mass Storage"
   "idVendor"=dword:066F
   "Manufacturer"="Freescale"
   "idProduct"=dword:37FF
   "Product"="UUT"
   "bcdDevice"=dword:0
   "SerialNumber"="0802270905553"
   "DeviceName"="NAND FLASH"
```

5.2 The process flow

Firstly, the host sends UUT image to the i.MX device. The client driver will call USB MSC Card Reader layer to recognize the i.MX device as a Card Reader.
Then host send commands and data to the Card Reader. The USB MSC Card Reader layer will receive the special command “SCSI_UTP” package and send it to Universal Transfer Protocol layer to check it is a command or data. Then the package will be transfer to Universal Client Engine layer to handle. If it is a command it will be parsed and dealt here. Or if it is data it will send with media driver.
The process flow shows as following.



5.3 HID mode implementation

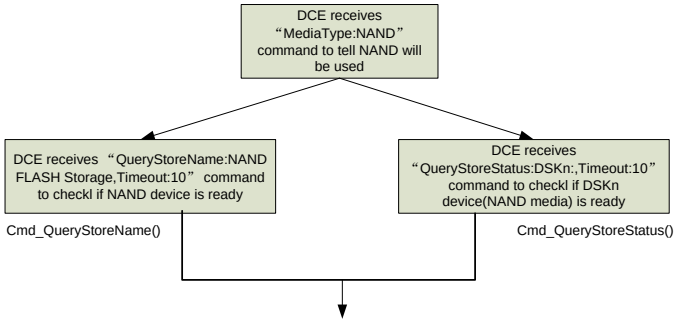
HID mode i.MX device includes i.MX233, i.MX28 and i.MX508. i.MX233 and i.MX28 will follow one process and i.MX508 will execute another process.

5.3.1 i.MX233/28 implementation

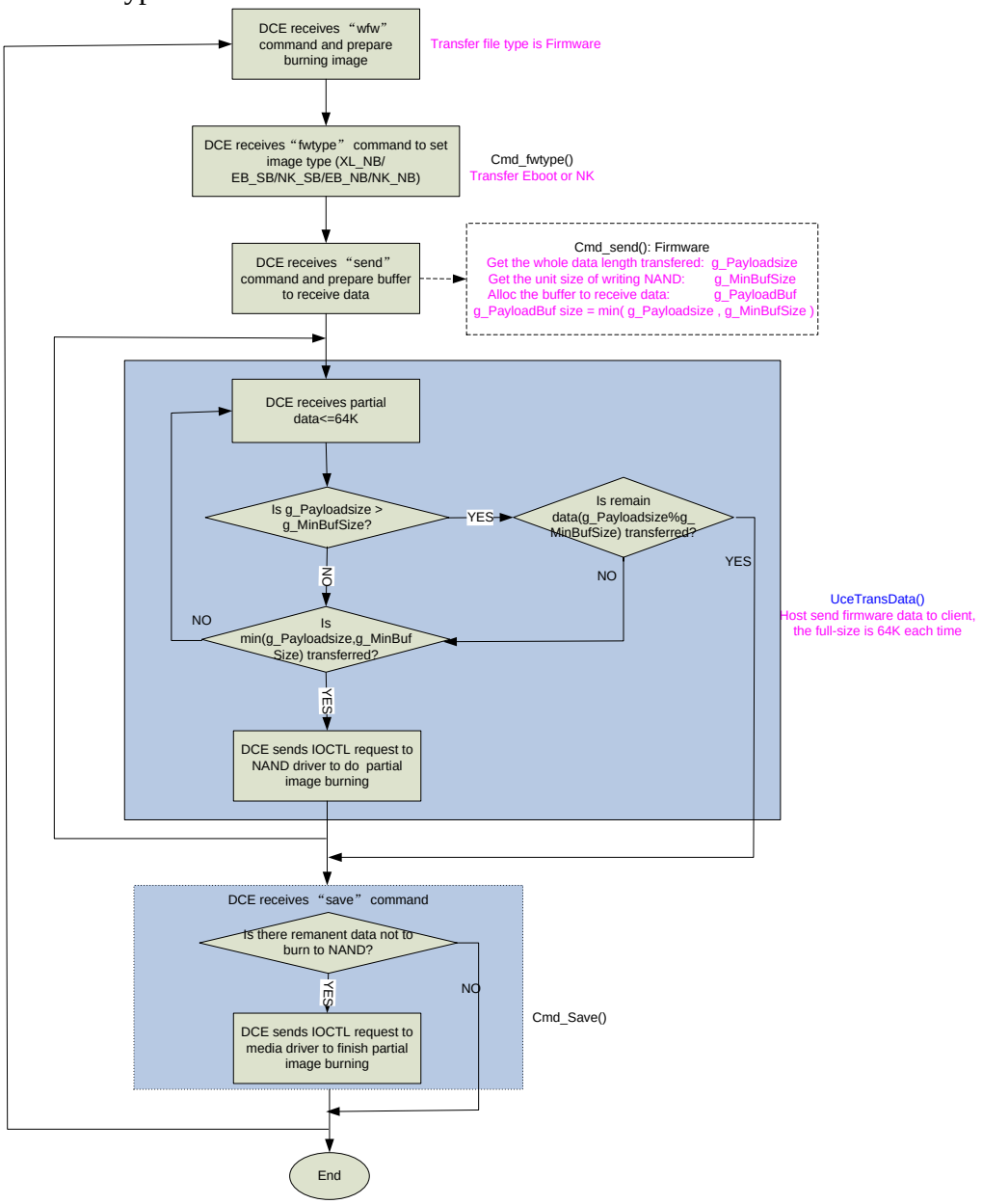
There are three file types: Firmware, Raw data and file data. Transferring xldr/eboot/nk image use either one of Firmware and Raw data type. And transferring general file uses file data type. Setting media type and checking media status firstly before receiving and burning data.

5.3.1.1 NAND operations implementation

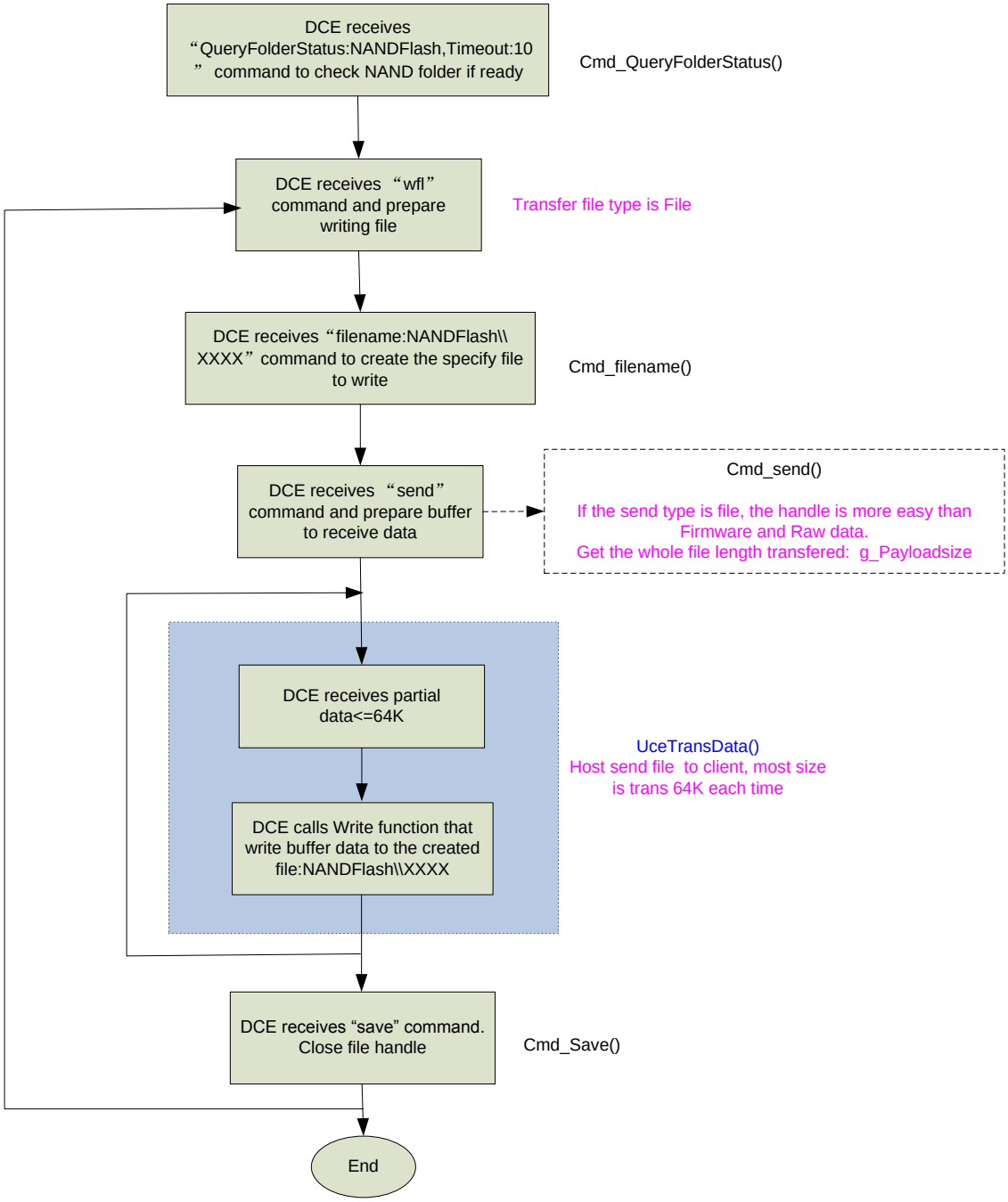
- Firstly, set NAND type and check NAND if it is ready for burning.



- Then the firmwares are sent and burned to media of NAND. i.MX233/28 uses Firmware type to transfer xldr/eboot/nk to NAND media.



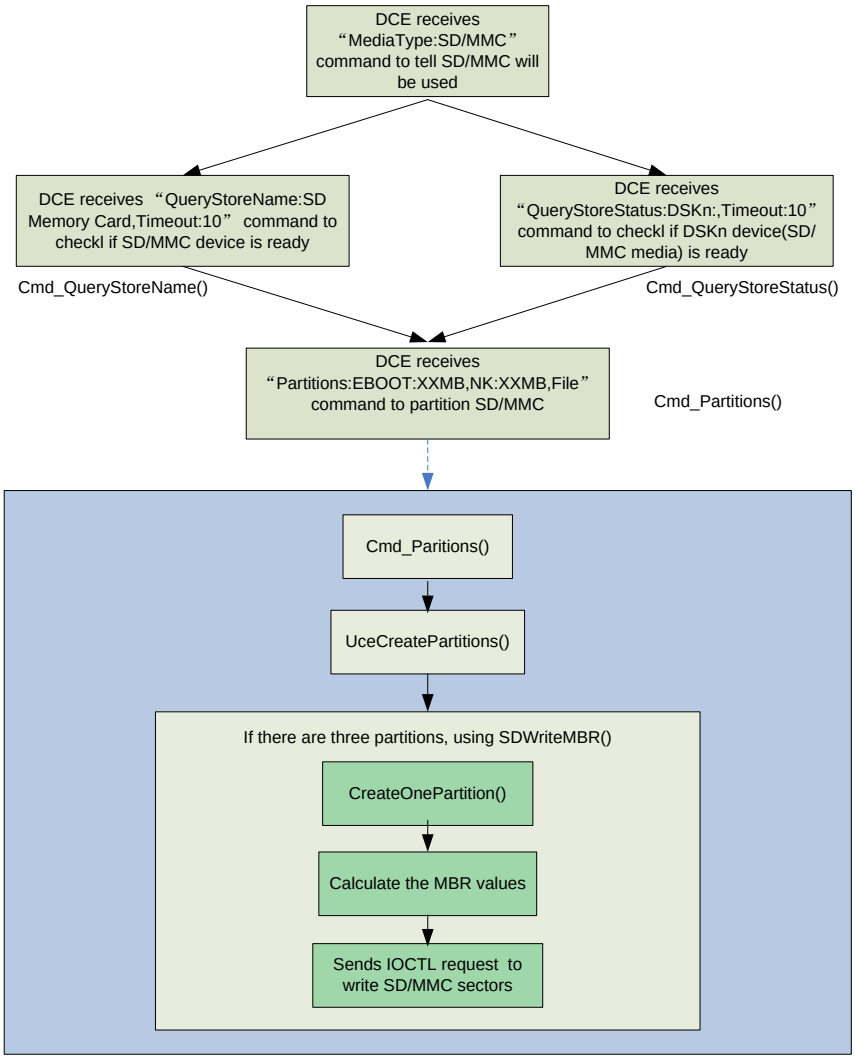
- The process of files transferred to NAND media shows as below.



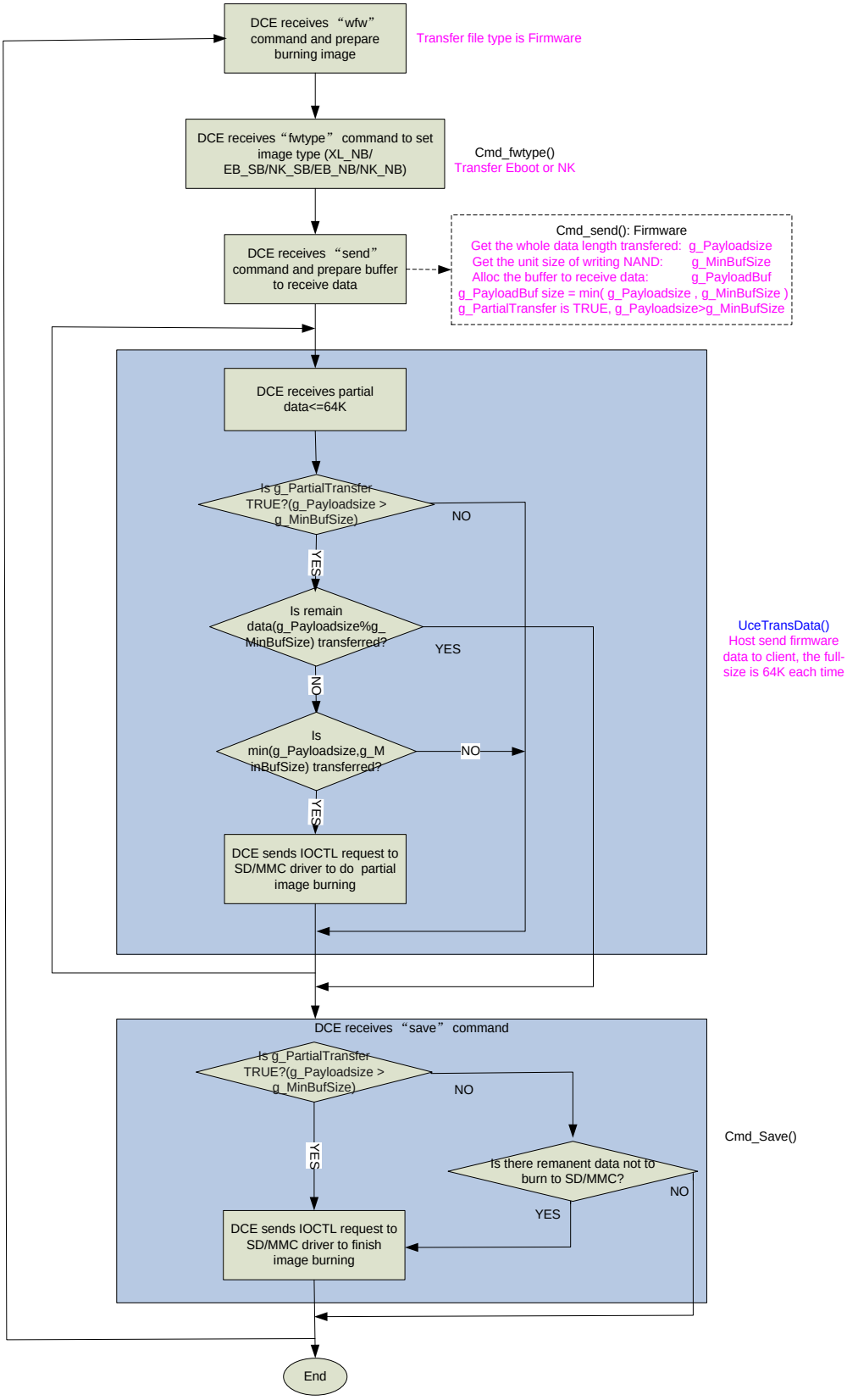
5.3.1.2 SD/MMC operations implementation

- Set SD/MMC media type and check its status.
  i.MX233/28 uses MBR mode to burn firmware. The MBR (master boot record) is the 512-byte boot sector that is the first sector.
  The SD/MMC is divided into three partitions: the first is the File partition, the second is Eboot partition and the three is the NK partition. These partitions size is

decided by the xml command. And all the partition information is recorded in MBR.
About the MBR knowledge, see *Master Boot Record WIKI*.
NOTE: If the SD/MMC isn't formatted to create a file system, it will not be capable for files writing. So add the function of CreateOnePartition before writing MBR is for writing files.



- The firmwares are sent and burned to media of SD/MMC.
  The most processes of SD/MMC are same with NAND's. But NAND always uses partial transfer.


DCE receives “wfw” command and prepare burning image
Transfer file type is Firmware
DCE receives “fwtype” command to set image type (XL_NB/ EB_SB/NK_SB/EB_NB/NK_NB)
Cmd_fwtype()
Transfer Eboot or NK
DCE receives “send” command and prepare buffer to receive data
Cmd_send(): Firmware
Get the whole data length transfered: g_Payloadsize
Get the unit size of writing NAND: g_MinBufSize
Alloc the buffer to receive data: g_PayloadBuf
g_PayloadBuf size = min(g_Payloadsize , g_MinBufSize)
g_PartialTransfer is TRUE, g_Payloadsize>g_MinBufSize
DCE receives partial data<=64K
Is g_PartialTransfer TRUE?(g_Payloadsize > g_MinBufSize)
NO
YES
Is remain data(g_Payloadsize%g_ MinBufSize) transferred?
YES
NO
Is min(g_Payloadsize,g_M inBufSize) transferred?
NO
YES
DCE sends IOCTL request to SD/MMC driver to do partial image burning
UceTransData()
Host send firmware data to client, the full-size is 64K each time
DCE receives “save” command
Is g_PartialTransfer TRUE?(g_Payloadsize > g_MinBufSize)
NO
YES
Is there remanent data not to burn to SD/MMC?
NO
YES
DCE sends IOCTL request to SD/MMC driver to finish image burning
Cmd_Save()
End

- The files are written to SD/MMC.
  The processes of files written are same with NAND's processes. Only the media type change from NAND to SD/MMC.
  The processes refer to chapter - *5.3.1.1: The process of files transferred to NAND media.*

5.3.1.3 Generate UUT image

I. Create a UUT project.
   Selects all modules need be used in catalog, such as NAND, SD, USB modules.

II. Set Environment variables
   There are two environment variables must be set: BSP_UUT and IMGUUT.
   - BSP_UUT is used for the UUT module built.
   - IMGUUT is used in config.bib to get a small image. The NK_SIZE is set by user based on the actual requirement. An example shows in below.

```
IF IMGUUT
      #define NK_SIZE     00300000    ;3MB
ENDIF
```

III. Set platform.reg and platform.bib
   Add UUT registry to platform.reg, see *5.1 – Register settings.*
   Add the following comments in platform.bin:

```
IF BSP_UUT
   UUT.dll    $(_FLATRELEASEDIR)\UUT.dll     NK  SHK
ENDIF ; BSP_UUT.
```

IV. Build the UUT project.
   It generates nk.sb image. Rename nk.sb as uce.sb. Then the UUT image is generated as uce.sb.

V. Copy uce.sb to corresponding folder, e.g. *"\mfgtool\Profiles\MXxx WinCE Update\OS firmware".*

5.3.2 i.MX508 implementation

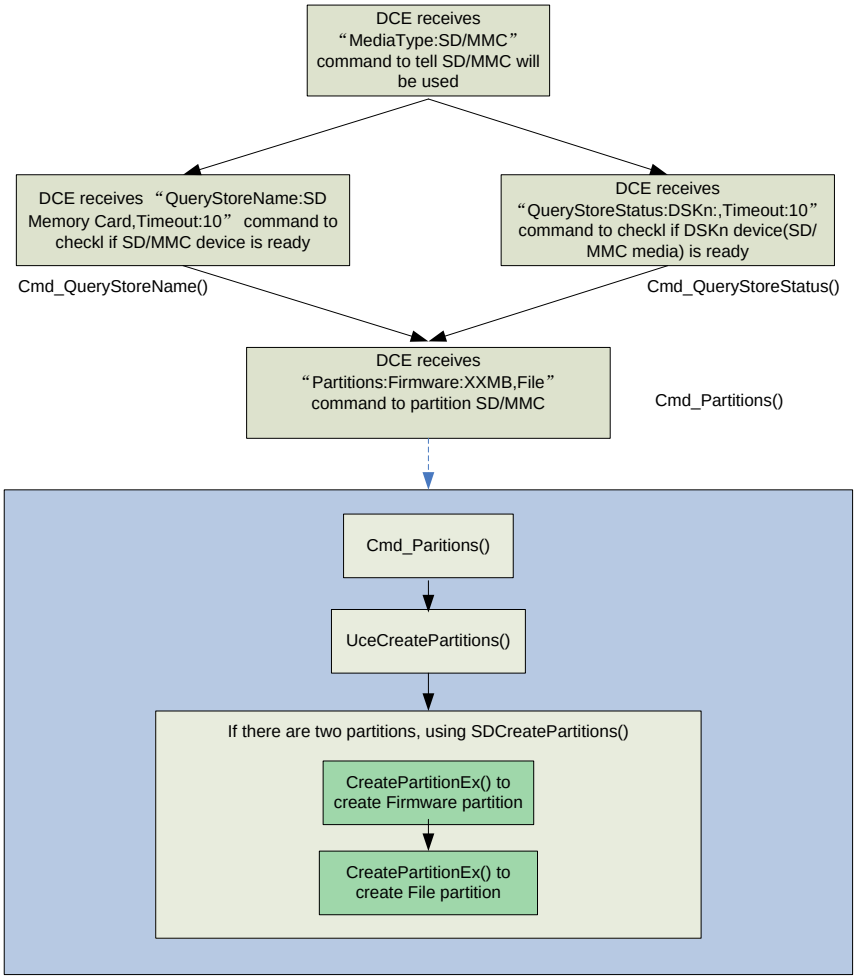
5.3.2.1 NAND operations implementation

N/A.

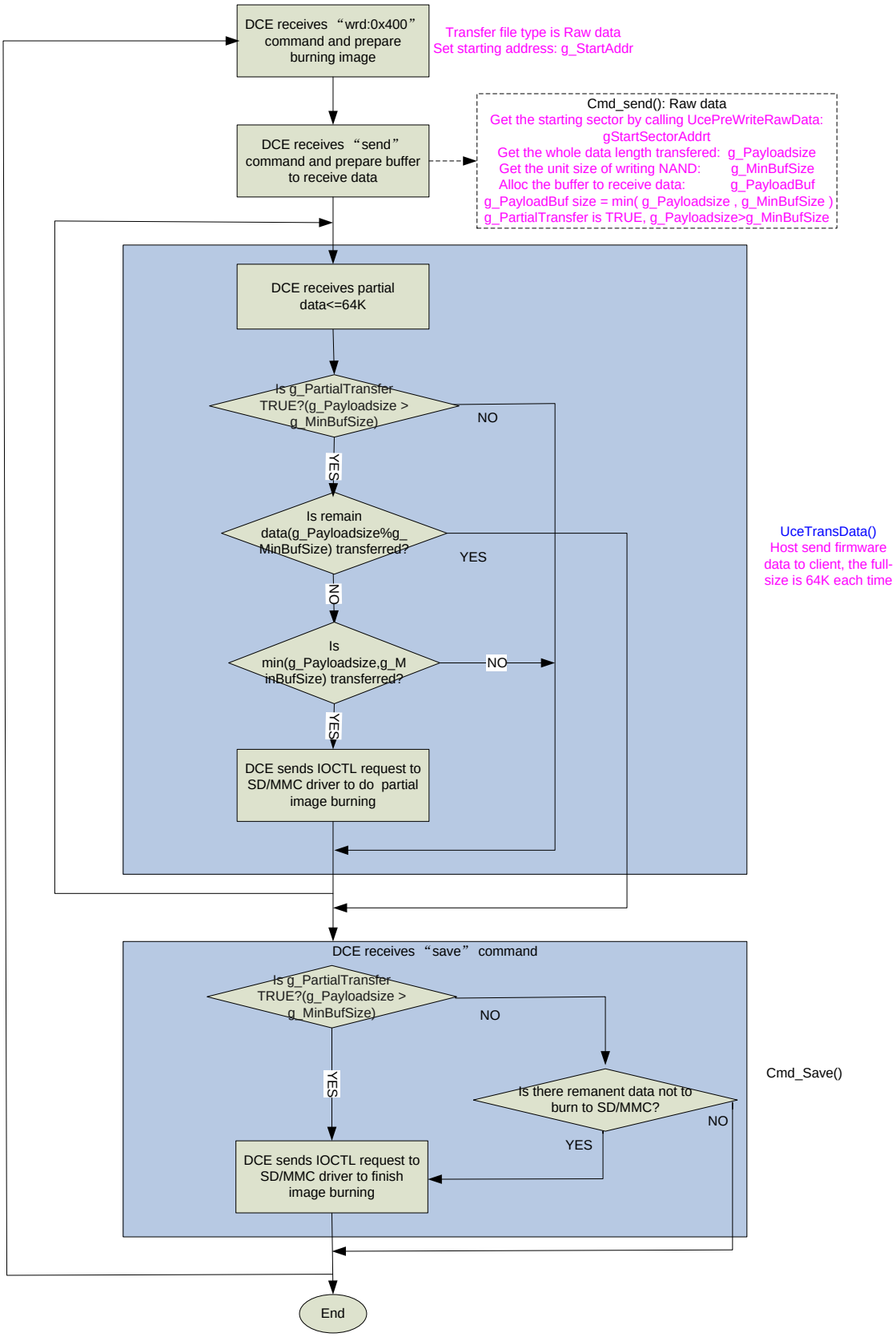
5.3.2.2 SD/MMC operations implementation

- Set SD/MMC media type and check SD/MMC status.

i.MX508 uses two partitions for SD/MMC: the first is Firmware partition and the second is File partition. The firmware partition uses to store xldr, eboot and nk image.



- The raw data are sent and burned to media of SD/MMC.


DCE receives “wrd:0x400” command and prepare burning image
Transfer file type is Raw data
Set starting address: g_StartAddr
DCE receives “send” command and prepare buffer to receive data
Cmd_send(): Raw data
Get the starting sector by calling UcePreWriteRawData: gStartSectorAddrt
Get the whole data length transfered: g_Payloadsize
Get the unit size of writing NAND: g_MinBufSize
Alloc the buffer to receive data: g_PayloadBuf
g_PayloadBuf size = min(g_Payloadsize , g_MinBufSize)
g_PartialTransfer is TRUE, g_Payloadsize>g_MinBufSize
DCE receives partial data<=64K
Is g_PartialTransfer TRUE?(g_Payloadsize > g_MinBufSize)
NO
YES
Is remain data(g_Payloadsize%g_MinBufSize) transferred?
YES
NO
Is min(g_Payloadsize,g_MinBufSize) transferred?
NO
YES
DCE sends IOCTL request to SD/MMC driver to do partial image burning
UceTransData()
Host send firmware data to client, the full-size is 64K each time
DCE receives “save” command
Is g_PartialTransfer TRUE?(g_Payloadsize > g_MinBufSize)
NO
YES
Is there remanent data not to burn to SD/MMC?
NO
YES
DCE sends IOCTL request to SD/MMC driver to finish image burning
Cmd_Save()
End

- Writing files to SD/MMC.
  The processes please refer to chapter - *5.3.1.1: The process of files transferred to NAND media.*

5.3.2.3 Generate corresponding image

I. Create a UUT project
   Selects all modules need be used in catalog.

II. Set Environment variables
   There are two environment variables must be set: BSP_UUT and IMGTINY.
   - BSP_UUT is used for the UUT module built.
   - IMGTINY is used in config.bib to make the image as small as possible by removing most modules. It is defined by Microsoft.
     NOTE: Suggest using an environment variables defined by user for the security such as IMGUUT.

III. Set platform.reg and platform.bib
   Add UUT registry to platform.reg, see *5.1 – Register settings.*
   Add the following comments in platform.bin:

```
IF BSP_UUT
   UUT.dll    $(_FLATRELEASEDIR)\UUT.dll     NK  SHK
ENDIF ; BSP_UUT.
```

IV. Get UUT image
   - Build the UUT project. It generates nk.nb0 image.
   - Rename nk.nb0 as uce.nb0. Then the UUT image is generated as uce.nb0.
   - Copy uce.nb0 to corresponding folder, e.g. *"\mfgtool\Profiles\MXxx WinCE Update\OS firmware".*

V. Get eboot_uut image
   It is for initialization such like kitl parameters setting.
   - Update main.c in *$(_TARGETPLATROOT)\SRC\BOOTLOADER\COMMON.*
   - Rebuild it and generate eboot.nb0
   - Rename eboot.nb0 to eboot_uut.nb0
   - Copy eboot_uut.nb0 to corresponding folder, e.g. *"\mfgtool\Profiles\MXxx WinCE Update\OS firmware".*

5.4 Bulk-IO mode implementation

i.MX35, i.MX51 and i.MX53 are bulk-IO mode i.MX device.

5.4.1 NAND operation implementation

Please refer to chapter - *5.3.1.1 NAND operations implementation.*

The process of UCE is same with 5.3.1.1. The difference is the operations how to control NAND media.

5.4.2 SD/MMC operation implementation

Please refer to chapter - *5.4.2.1 SD/MMC operations implementation.*

The process of UCE is same with 5.4.2.1. And the difference is in the operations how to control SD/MMC media.

5.4.3 Generate corresponding image

There are two images shown as below.

- UUT image – uce.nb0
- Eboot_uut image – eboot_uut.nb0

Please refer to chapter - *5.3.2.3 Generate corresponding image* to generate the two images.