Qualcomm Technologies, Inc.

# GUID Partition Tables and Programming

## Application Note

80-N7350-1 C

May 21, 2015

Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

# Revision history

| Revision | Date | Description |
|---|---|---|
| A | Aug 2011 | Initial release |
| B | Feb 2013 | Added Table 2-4, Section 2.1.4, and Section 2.2; numerous changes in Chapter 4 |
| C | May 2015 | Added Chapter 3, Section 5.1.1, and Section 5.2.1. |

# Contents

## Figures

## Tables

# 1 Introduction

## 1.1 Purpose

This document provides an introduction to the Globally Unique Identifier (GUID) Partition Table (GPT), as well as a guideline on how to program the image into devices using Qualcomm QPST tools.

## 1.2 Conventions

Function declarations, function names, type declarations, attributes, and code samples appear in a different font, for example, `#include`.

Code variables appear in angle brackets, for example, `<number>`.

Commands to be entered appear in a different font, for example, **`copy a:*.* b:`**.

Button and key names appear in bold font, for example, click **Save** or press **Enter**.

Shading indicates content that has been added or changed in this revision of the document.

## 1.3 Technical assistance

For assistance or clarification on information in this document, submit a case to Qualcomm Technologies, Inc. (QTI) at https://support.cdmatech.com/.

If you do not have access to the CDMATech Support website, register for access or send email to support.cdmatech@qti.qualcomm.com.

# 2 GPT partitions

GPT is a standard for the layout of the partition table on a physical storage device. Although it forms a part of the Extensible Firmware Interface (EFI) standard (the proposed replacement by Intel for the PC BIOS), it is also used on some BIOS systems and embedded systems.

GPT partitioning was created as a replacement for MBR partitioning, primarily to overcome its size limitation. While an MBR partition can handle up to 2 TB (2048 GB), a GPT partition can handle up to 9.4 ZB (zettabytes), for example, 8589934592 TB.

GPT also adds enhanced security, features, and redundancy into the partitioning scheme.

## 2.1 GPT partition scheme

GPTs use Logical Block Addressing (LBA) in place of the historical Cylinder-Head-Sector (CHS) addressing.

The partition table information is stored in the GPT header. However, to maintain compatibility, GPT retains the MBR entry as the first sector, followed by the primary partition table header that takes up exactly one sector (see Figure 2-1).

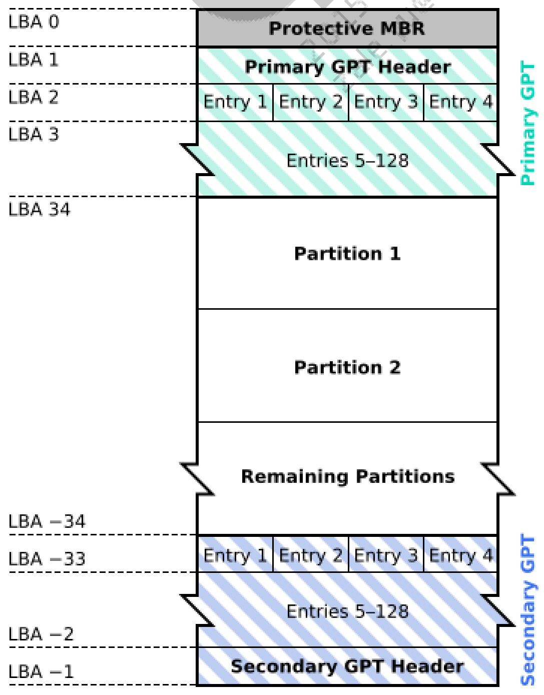


**Figure 2-1 GPT partition scheme**

This is followed by 128 partition entries that take up exactly 32 sectors.

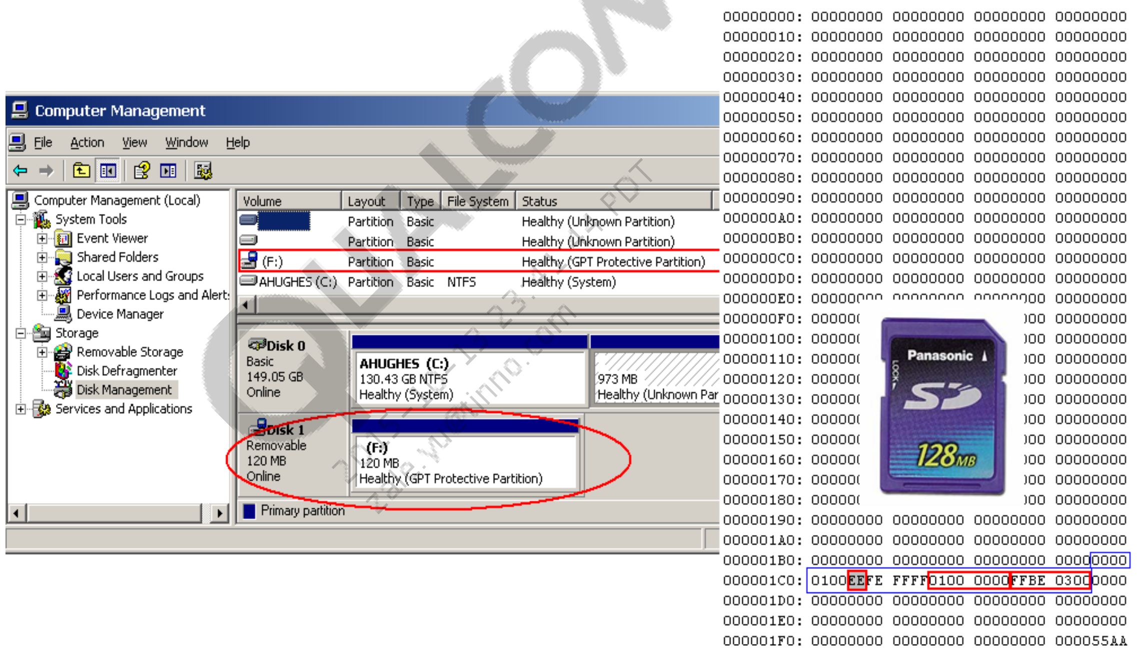
Then, for safety (redundancy), the header and partition entries are duplicated at the end of the disk.

For example, on a 1000-sector disk, the backup GPT header is at sector 999 and the backup partition array begins at sector 967 and ends at 998.

Each GPT header has an entry pointing to the location of the other GPT header.

## 2.1.1 Legacy MBR (LBA 0)

In the GPT specification, the location corresponding to the MBR in an MBR-based disk is structured in a way that prevents MBR-based disk utilities from misrecognizing, and possibly overwriting, GPT disks. This is referred to as a protective MBR (Figure 2-2).



| Volume | Layout | Type | File System | Status |
|---|---|---|---|---|
| | Partition | Basic | | Healthy (Unknown Partition) |
| | Partition | Basic | | Healthy (Unknown Partition) |
| (F:) | Partition | Basic | | Healthy (GPT Protective Partition) |
| AHUGHES (C:) | Partition | Basic | NTFS | Healthy (System) |

Disk 0
Basic
149.05 GB
Online

AHUGHES (C:)
130.43 GB NTFS
Healthy (System)

973 MB
Healthy (Unknown Par

Disk 1
Removable
120 MB
Online

(F:)
120 MB
Healthy (GPT Protective Partition)

```
00000000: 00000000 00000000 00000000 00000000
00000010: 00000000 00000000 00000000 00000000
00000020: 00000000 00000000 00000000 00000000
00000030: 00000000 00000000 00000000 00000000
00000040: 00000000 00000000 00000000 00000000
00000050: 00000000 00000000 00000000 00000000
00000060: 00000000 00000000 00000000 00000000
00000070: 00000000 00000000 00000000 00000000
00000080: 00000000 00000000 00000000 00000000
00000090: 00000000 00000000 00000000 00000000
000000A0: 00000000 00000000 00000000 00000000
000000B0: 00000000 00000000 00000000 00000000
000000C0: 00000000 00000000 00000000 00000000
000000D0: 00000000 00000000 00000000 00000000
000000E0: 00000000 00000000 00000000 00000000
000000F0: 00000                   00 00000000
00000100: 00000                   00 00000000
00000110: 00000                   00 00000000
00000120: 00000                   00 00000000
00000130: 00000                   00 00000000
00000140: 00000                   00 00000000
00000150: 00000                   00 00000000
00000160: 00000                   00 00000000
00000170: 00000                   00 00000000
00000180: 00000                   00 00000000
00000190: 00000000 00000000 00000000 00000000
000001A0: 00000000 00000000 00000000 00000000
000001B0: 00000000 00000000 00000000 00000000
000001C0: 0100EEFE FFFF0100 0000FFBE 03000000
000001D0: 00000000 00000000 00000000 00000000
000001E0: 00000000 00000000 00000000 00000000
000001F0: 00000000 00000000 00000000 000055AA
```

**Figure 2-2 Protective MBR (LBA 0)**

A single partition type of 0xEE, encompassing the entire GPT drive, is indicated and identified as GPT. Operating systems (OS) that cannot read GPT disks recognize the disk as containing one partition of an unknown type and no empty space, and typically refuse to modify the disk unless the user explicitly requests and confirms deletion of this partition. This minimizes accidental erasures. Furthermore, GPT-aware OSs check the protective MBR, and if the enclosed partition type is not of type 0xEE or if there are multiple partitions defined on the target device, the device must not be manipulated.

## 2.1.2 Partition table header (LBA 1)

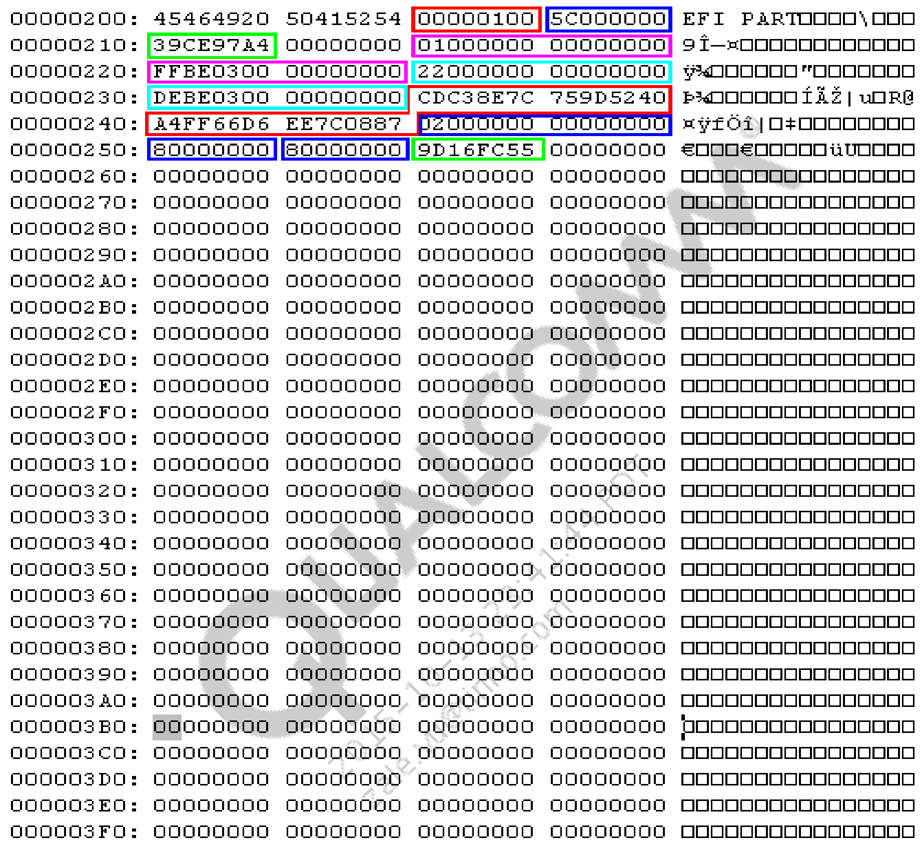
The partition table header defines the usable blocks on the disk. It also defines the number and size of the partition entries that make up the partition table. There are 128 partitions that can be created, each 128 bytes long. See Table 2-1.

**Table 2-1 Partition table header format**

| Offset | Length | Contents |
|---|---|---|
| 0 | 8 bytes | Signature (EFI PART, 45 46 49 20 50 41 52 54) |
| 8 | 4 bytes | Revision for GPT ver 1.0 (through at least UEFI ver 2.3.1), the value is 00 00 01 00) |
| 12 | 4 bytes | Header size in little endian (in bytes, usually 5C 00 00 00, meaning 92 bytes) |
| 16 | 4 bytes | CRC32 of header (0 to header size), with this field zeroed during calculation |
| 20 | 4 bytes | Reserved; must be zero |
| 24 | 8 bytes | Current LBA (location of this header copy) |
| 32 | 8 bytes | Backup LBA (location of the other header copy) |
| 40 | 8 bytes | First usable LBA for partitions (primary partition table last LBA + 1) |
| 48 | 8 bytes | Last usable LBA (secondary partition table first LBA - 1) |
| 56 | 16 bytes | Disk GUID (also referred as UUID on UNIXes) |
| 72 | 8 bytes | Partition entries starting LBA (always 2 in primary copy) |
| 80 | 4 bytes | Number of partition entries |
| 84 | 4 bytes | Size of a partition entry (usually 128) |
| 88 | 4 bytes | CRC32 of partition array |
| 92 | * | Reserved; must be 0s for the rest of the block (420 bytes for a 512-byte LBA) |

The header contains the disk GUID. It records its own size and location (always LBA 1) and the size and location of the secondary GPT header and table (always the last sectors on the disk). It also contains a CRC32 checksum for itself and for the partition table, which can be verified by the firmware, bootloader, and/or OS on boot. Because of this, HEX editors must not be used to modify the contents of the GPT. Such modification renders the checksum invalid. In this case, the primary GPT might be overwritten with the secondary GPT by disk recovery software. If both GPTs contain invalid checksums, the disk is unusable.

Figure 2-3 shows an actual header that consists of 92 bytes of useful information. The rest are zero-padded, for example, 420 bytes of 0s. Green highlighting in the figure shows the two CRCs. One protects the header, and the other protects the partition array. Editing this header with a HEX editor is useless without some way of calculating the CRCs.



```
00000200: 45464920 50415254 00000100 5C000000 EFI PART□□□□\□□□
00000210: 39CE97A4 00000000 01000000 00000000 9Î—¤□□□□□□□□□□□□
00000220: FFBE0300 00000000 22000000 00000000 ÿ¾□□□□□□"□□□□□□□
00000230: DEBE0300 00000000 CDC38E7C 759D5240 Þ¾□□□□□□ÍÃŽ|u□R@
00000240: A4FF66D6 EE7C0887 02000000 00000000 ¤ÿfÖî|□‡□□□□□□□□
00000250: 80000000 80000000 9D16FC55 00000000 €□□□€□□□□□üU□□□□
00000260: 00000000 00000000 00000000 00000000 □□□□□□□□□□□□□□□□
00000270: 00000000 00000000 00000000 00000000 □□□□□□□□□□□□□□□□
00000280: 00000000 00000000 00000000 00000000 □□□□□□□□□□□□□□□□
00000290: 00000000 00000000 00000000 00000000 □□□□□□□□□□□□□□□□
000002A0: 00000000 00000000 00000000 00000000 □□□□□□□□□□□□□□□□
000002B0: 00000000 00000000 00000000 00000000 □□□□□□□□□□□□□□□□
000002C0: 00000000 00000000 00000000 00000000 □□□□□□□□□□□□□□□□
000002D0: 00000000 00000000 00000000 00000000 □□□□□□□□□□□□□□□□
000002E0: 00000000 00000000 00000000 00000000 □□□□□□□□□□□□□□□□
000002F0: 00000000 00000000 00000000 00000000 □□□□□□□□□□□□□□□□
00000300: 00000000 00000000 00000000 00000000 □□□□□□□□□□□□□□□□
00000310: 00000000 00000000 00000000 00000000 □□□□□□□□□□□□□□□□
00000320: 00000000 00000000 00000000 00000000 □□□□□□□□□□□□□□□□
00000330: 00000000 00000000 00000000 00000000 □□□□□□□□□□□□□□□□
00000340: 00000000 00000000 00000000 00000000 □□□□□□□□□□□□□□□□
00000350: 00000000 00000000 00000000 00000000 □□□□□□□□□□□□□□□□
00000360: 00000000 00000000 00000000 00000000 □□□□□□□□□□□□□□□□
00000370: 00000000 00000000 00000000 00000000 □□□□□□□□□□□□□□□□
00000380: 00000000 00000000 00000000 00000000 □□□□□□□□□□□□□□□□
00000390: 00000000 00000000 00000000 00000000 □□□□□□□□□□□□□□□□
000003A0: 00000000 00000000 00000000 00000000 □□□□□□□□□□□□□□□□
000003B0: 00000000 00000000 00000000 00000000 □□□□□□□□□□□□□□□□
000003C0: 00000000 00000000 00000000 00000000 □□□□□□□□□□□□□□□□
000003D0: 00000000 00000000 00000000 00000000 □□□□□□□□□□□□□□□□
000003E0: 00000000 00000000 00000000 00000000 □□□□□□□□□□□□□□□□
000003F0: 00000000 00000000 00000000 00000000 □□□□□□□□□□□□□□□□
```

**Figure 2-3 Primary GPT header**

### 2.1.3 Partition entries (LBA 2 to 33)

The GPT uses simple and straightforward entries to describe partitions. The first 16 bytes designate the partition type GUID, for example, the GUID for an EFI system partition is {C12A7328-F81F-11D2-BA4B-00A0C93EC93B}. The second 16 bytes contain a GUID unique to the partition. Starting and ending 64-bit LBAs are also recorded here, and space is allocated for partition names and attributes.

**Table 2-2 GPT partition entry format**

| Offset | Length | Contents |
|---|---|---|
| 0 | 16 bytes | Partition type GUID |
| 16 | 16 bytes | Unique partition GUID |
| 32 | 8 bytes | First LBA (little-endian) |
| 40 | 8 bytes | Last LBA (inclusive, usually odd) |
| 48 | 8 bytes | Attribute flags, for example, bit 60 denotes read-only |
| 56 | 72 bytes | Partition name (36 UTF-16LE code units) |

The partition type GUID is a specific 16-byte value that indicates the type of partition.

**Table 2-3 Partition GUID type on the A-family chip BSP**

| Partition | GUID type |
|---|---|
| Unused entry | 00000000-0000-0000-0000-000000000000 |
| SBL1 | DEA0BA2C-CBDD-4805-B4F9-F428251C3E98 |
| SBL2 | 8C6B52AD-8A9E-4398-AD09-AE916E53AE2D |
| SBL3 | 05E044DF-92F1-4325-B69E-374A82E97D6E |
| APPSBL | 400FFDCD-22E0-47E7-9A23-F16ED9382388 |
| TZ | A053AA7F-40B8-4B1C-BA08-2F68AC71A4F4 |
| RPM | 098DF793-D712-413D-9D4E-89D711772228 |
| Modem_FS1 | EBBEADAF-22C9-E33B-8F5D-0E81686A68CB |
| Modem_FS2 | 0A288B1F-22C9-E33B-8F5D-0E81686A68CB |
| Modem_FSG | 638FF8E2-22C9-E33B-8F5D-0E81686A68CB |
| Modem_Fusion_FS1 | 2290BE64-22C9-E33B-8F5D-0E81686A68CB |
| Modem_Fusion_FS2 | 346C26D1-22C9-E33B-8F5D-0E81686A68CB |
| Modem_Fusion_FSG | BF64FB9C-22C9-E33B-8F5D-0E81686A68CB |

**Table 2-4 Partition GUID types additional to the B-family chip BSP**

| Partition | GUID type |
|---|---|
| DBI | 00000000-0000-0000-0000-000000000000 |
| DDR | C12A7328-F81F-11D2-BA4B-00A0C93EC93B |

Figure 2-4 is an example of three partition entries decoded. The size is the same, 7.8 MB, and the type GUID is the same, FAT. However, they all have unique partition GUIDs.

```
$PartitionTypeGUID       0xC79926B7B668C0874433B9E5EBD0A0A2
$UniquePartitionGUID     0x373C17CF53BC7FB149B85A927ED24483
$FirstLBA                0x0000000000000022 (34)
$LastLBA                 0x0000000000003EC0 (16064)      Size is 7.83 MiB
$AttributeFlags          0x0000000000000000 (0)
$PartitionName
0x0000000000000000000000000000000000000000000000000000000000000000000

$PartitionTypeGUID       0xC79926B7B668C0874433B9E5EBD0A0A2
$UniquePartitionGUID     0x1D3C4663FC172F904EC7E0C7A8CF84EC
$FirstLBA                0x0000000000003EC1 (16065)
$LastLBA                 0x0000000000007D81 (32129)      Size is 7.84 MiB
$AttributeFlags          0x0000000000000000 (0)
$PartitionName
0x0000000000000000000000000000000000000000000000000000000000000000000

$PartitionTypeGUID       0xC79926B7B668C0874433B9E5EBD0A0A2
$UniquePartitionGUID     0x04A9B2AAEF96DAAE465F429D0EF5C6E2
$FirstLBA                0x0000000000007D82 (32130)
$LastLBA                 0x000000000000BC42 (48194)      Size is 7.84 MiB
$AttributeFlags          0x0000000000000000 (0)
$PartitionName
0x0000000000000000000000000000000000000000000000000000000000000000000
```

**Figure 2-4 Sample partition entries**

## 2.1.4 Parser instructions

The following values are parameters that are passed to the parser. They do not contain partition information but do affect the output files that are generated.

- WRITE_PROTECT_BOUNDARY_IN_KB – Specifies the write protect size in KBs that must be aligned to when write protection is enabled. The value depends on the eMMC device that is used. Check the eMMC data sheet to determine the write protect group size that is supported.
- GROW_LAST_PARTITION_TO_FILL_DISK – When set to TRUE, this parameter sets the last partition listed in the physical partition grow to fill the remainder of the disk when it is programmed. If set to FALSE, the partitions are written with the exact size specified in the partition table. If all of the space was not used, the extra space remains unpartitioned and blank.

## 2.2 Partition customization

Licensees often want to customize and add their own partitions to partition.xml. Be aware of the following when partitions are added:

- Many Android™ builds are highly dependent on partition order. Be careful when partitions are added in the middle of the partition.xml file.
- If no file is specified for the partition, nothing is written to that partition. If there was data in that partition before, it still exists there. To remove the data in the partition, load a blank file.
- If GROW_LAST_PARTITION_TO_FILL_DISK = TRUE, the last partition listed in each physical partition is expanded to fill the remainder of that physical partition. Do not add a partition as the last partition unless that partition is intended to fill the disk.
- Be sure the partition type is defined correctly. Do not simply use any partition type, and specifically do not use partition types used by boot loaders.
- Be sure to change WRITE_PROTECT_BOUNDARY_IN_KB to match the size of the eMMC device. The value is chosen to be the maximum in the reference file.

## 2.3 How PBL detects a bootable GPT partition

The PBL first attempts to detect the primary GPT partition table. If this fails, it detects the backup GPT partition table.

The right-hand side of the chart details how this detection works. The PBL reads the GPT headers from the specified sector and validates the signature, start sector number, number of sectors, and header CRC. If detection passes, the PBL reads the partition entry table according to the GPT header and calculates the entry table CRC. Detection of the GPT partition fails if any of the preceding steps fail.

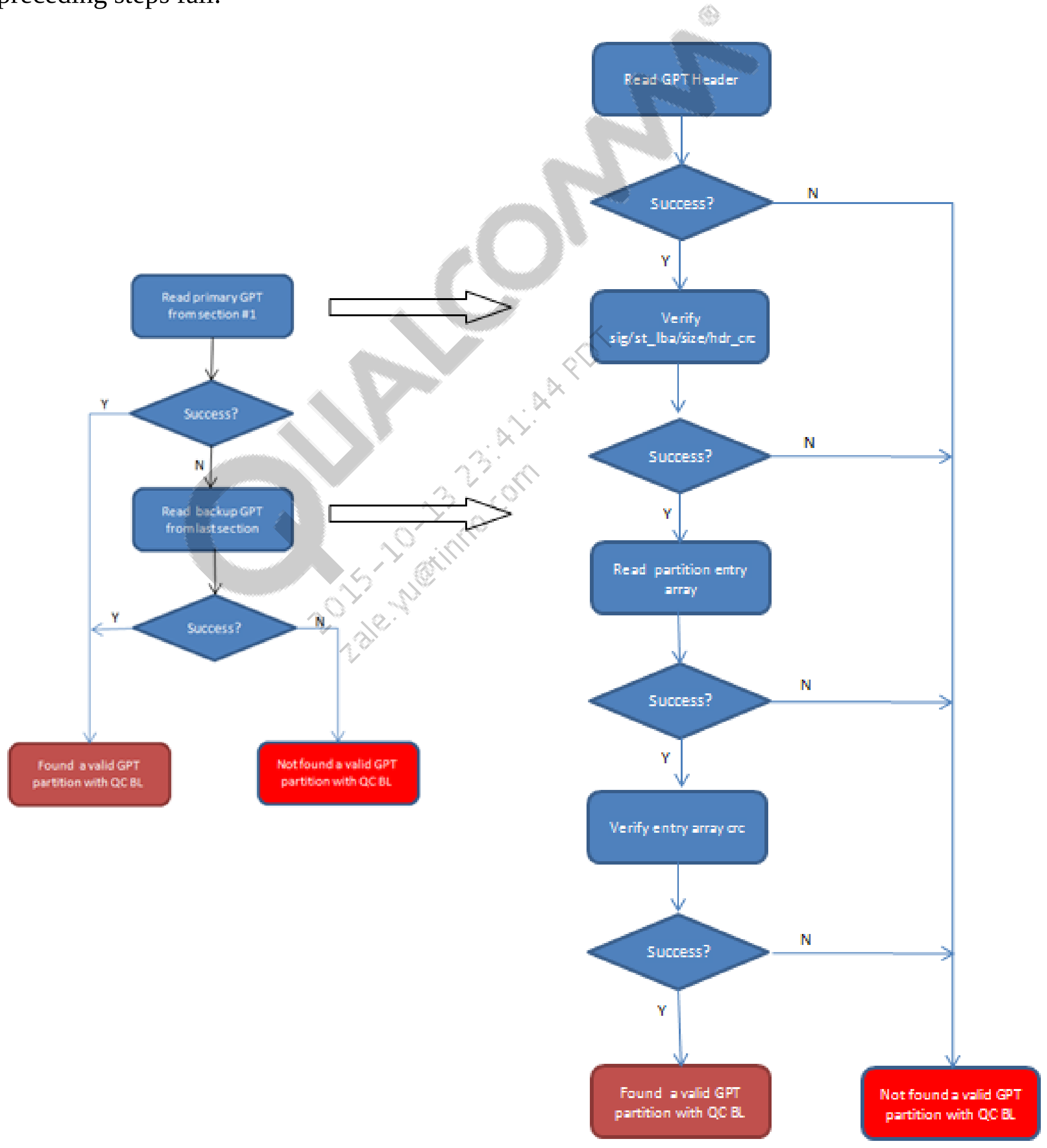

Read GPT Header
Success?
N
Y
Verify sig/st_lba/size/hdr_crc
Success?
N
Y
Read partition entry array
Success?
N
Y
Verify entry array crc
Success?
N
Y
Found a valid GPT partition with QC BL
Not found a valid GPT partition with QC BL
Read primary GPT from section #1
Success?
Y
N
Read backup GPT from last section
Success?
Y
N
Found a valid GPT partition with QC BL
Not found a valid GPT partition with QC BL

**Figure 2-5 GPT detection in PBL**

# 3 UFS partitioning

NOTE: This chapter was added to this document revision.

## 3.1 UFS provisioning

UFS must be provisioned before using. The OEM must change the provisioned XML and make sure that it fits customer UFS components.

The following is an example of provisioning XML:

```
<?xml version="1.0" ?>
<data>
    <ufs bNumberLU="6" bBootEnable="1" bDescrAccessEn="0" bInitPowerMode="1
" bHighPriorityLUN="0x5" bSecureRemovalType="0" bInitActiveICCLevel="0" wPe
riodicRTCUpdate="0" bConfigDescrLock="1"/>

    <ufs LUNum="0" bLUEnable="1" bBootLunID="0" size_in_kb="24621056" bData
Reliability="0" bLUWriteProtect="0" bMemoryType="0" bLogicalBlockSize="0x0c
" bProvisioningType="2" wContextCapabilities="0" desc="LU 0 - User LUN" />
    <ufs LUNum="1" bLUEnable="1" bBootLunID="1" size_in_kb="8192"     bData
Reliability="0" bLUWriteProtect="1" bMemoryType="3" bLogicalBlockSize="0x0c
" bProvisioningType="2" wContextCapabilities="0" desc="LU 1 - Boot LUN A-
4MB" />
    <ufs LUNum="2" bLUEnable="1" bBootLunID="2" size_in_kb="8192"     bData
Reliability="0" bLUWriteProtect="1" bMemoryType="3" bLogicalBlockSize="0x0c
" bProvisioningType="2" wContextCapabilities="0" desc="LU 2 - Boot LUN B-
4MB" />
    <ufs LUNum="3" bLUEnable="1" bBootLunID="0" size_in_kb="131072"   bData
Reliability="0" bLUWriteProtect="1" bMemoryType="0" bLogicalBlockSize="0x0c
" bProvisioningType="2" wContextCapabilities="0" desc="LU 3 - OTP LUN-
128MB" />
    <ufs LUNum="4" bLUEnable="1" bBootLunID="0" size_in_kb="4194304"  bData
Reliability="0" bLUWriteProtect="1" bMemoryType="0" bLogicalBlockSize="0x0c
" bProvisioningType="2" wContextCapabilities="0" desc="LU 4 - Protected
Read-only LUN-512MB" />
    <ufs LUNum="5" bLUEnable="1" bBootLunID="0" size_in_kb="1572864"  bData
Reliability="0" bLUWriteProtect="0" bMemoryType="0" bLogicalBlockSize="0x0c
" bProvisioningType="2" wContextCapabilities="0" desc="LU 5 - Protected
Read-write LUN-2.5GB" />
    <ufs LUNum="6" bLUEnable="0" bBootLunID="0" size_in_kb="0"        bData
Reliability="0" bLUWriteProtect="0" bMemoryType="0" bLogicalBlockSize="0x0c
```

```
" bProvisioningType="0" wContextCapabilities="0" desc="LU 6 - Place Holder1
LUN" />
    <ufs LUNum="7" bLUEnable="0" bBootLunID="0" size_in_kb="0"        bData
Reliability="0" bLUWriteProtect="0" bMemoryType="0" bLogicalBlockSize="0x0c
" bProvisioningType="0" wContextCapabilities="0" desc="LU 7 - Place Holder2
LUN" />

    <ufs commit="1" />

</data>
```

The OEM can refer to *Universal Flasg Storage (UFS) Unified Memory Extention* [JESD220-1] to understand the meaning of the descriptors such as bLUWriteProtect, bBootLunID, and bMemoryType. Check with the UFS vendor to confirm which value to choose.

UFS provisioning is a one-time step and the device must be power cycled after provisioning. Usually the UFS device cannot be provisioned more than once. The OEM must check with the UFS vendor whether reprovisioning is required.

For example, there are two separate BOOT-LUNs (BOOT A and BOOT B), one of them is set as Well known BOOT (W-BOOT) LUN during the image download process (using JTAG tools or the QFIL/device programmer). During system boot, PBL looks for W-BOOT LUN and loads the XBL image from it.

**NOTE:** UFS Lun0 size configuration depends on the UFS card size minus the current allocated partition size.

**NOTE:** Regarding bConfigDescrLock="1", the provision is a one time operation in Samsung devices and this configuration parameter must be to set to one during provisioning. On Toshiba, the part can be reprovisioned multiple times as long as bConfigDescrLock="0"

**NOTE:** Regarding bSecureRemovalType="0", the Samsung part performs the data removal from the device by remapping its page tables (not by physical erase). On Toshiba SecureRemovalType="0" triggers an erase in the NAND, therefore it can take up to 10 min to complete any unmap command. Toshiba advises using bSecureRemovalType="3" for this configuration. A different bMemoryType value affects the initialization time of the memory. The value is chosen for the best performance for the specific manufacturer. Check with each vendor for their recommendation.

Refer to the *Qualcomm Flash Image Loader (QFIL) User Guide* [80-NN120-1] for more information on UFS provisioning with the QFIL tool.

## 3.2 UFS partitions

The UFS partition table is in the defined metabuild:

`common\config\ufs\partition.xml`

The following is an example of a UFS partition table:

```
    <physical_partition>
        <partition label="ssd" size_in_kb="8" type="2C86E742-745E-4FDD-
BFD8-B6A7AC638772" bootable="false" readonly="false" filename="" />
        <partition label="persist" size_in_kb="32768" type="6C95E238-E343-
4BA8-B489-8681ED22AD0B" bootable="false" readonly="false"
filename="persist.img" sparse="true"/>
        <partition label="cache" size_in_kb="262144" type="5594C694-C871-
4B5F-90B1-690A6F68E0F7" bootable="false" readonly="false"
filename="cache.img" sparse="true"/>
        <partition label="misc" size_in_kb="1024" type="82ACC91F-357C-4A68-
9C8F-689E1B1A23A1" bootable="false" readonly="false" filename="" />
        <partition label="keystore" size_in_kb="512" type="DE7D4029-0F5B-
41C8-AE7E-F6C023A02B33" bootable="false" readonly="false" filename="" />
        <partition label="devcfg" size_in_kb="128" type="F65D4B16-343D-
4E25-AAFC-BE99B6556A6D" bootable="false" readonly="false"
filename="devcfg.mbn"/>
        <partition label="userdata" size_in_kb="12582912" type="1B81E7E6-
F50D-419B-A739-2AEEF8DA3335" bootable="false" readonly="false"
filename="userdata.img" sparse="true"/>
    </physical_partition>

    <!-- This is LUN 1 - Boot LUN A" -->
    <physical_partition>
        <partition label="xbl" size_in_kb="0" type="DEA0BA2C-CBDD-4805-
B4F9-F428251C3E98" bootable="false" readonly="true" filename="xbl.elf"/>
    </physical_partition>

    <!-- This is LUN 2 - Boot LUN B" -->
    <physical_partition>
        <partition label="xbl" size_in_kb="0" type="DEA0BA2C-CBDD-4805-
B4F9-F428251C3E98" bootable="false" readonly="true" filename="xbl.elf"/>
    </physical_partition>

    <!-- This is LUN 3 - OTP LUN" -->
    <physical_partition>
        <partition label="ddr" size_in_kb="32" type="20A0C19C-286A-42FA-
9CE7-F64C3226A794" bootable="false" readonly="true"/>
        <partition label="cdt" size_in_kb="1" type="A19F205F-CCD8-4B6D-
8F1E-2D9BC24CFFB1" bootable="false" readonly="true"/>
        <partition label="last_parti" size_in_kb="0" type="00000000-0000-
0000-0000-000000000000" bootable="false" readonly="true" filename="" />
```

```
        </physical_partition>
…
```

**NOTE:** Devcfg is the device onfiguration parameter for ARM TrustZone 4.0 or higher.

**Table 3-1 UFS partition**

| LUN sequence | Partition purpose | Partitions | Property |
|---|---|---|---|
| LUN0 | User | HLOS Kernel, HLOS file system | Read/write |
| LUN1 | BOOT A (0x30) | XBL | Power on write protection |
| LUN2 | BOOT B (0x30) | XBL | Power on write protection |
| LUN3 | OTP | CDT, DDR | Power on write protection |
| LUN4 | Protected RO area | RPM,TZ, SDI, UEFI,ACPI,LK,FSG,UFI Vars, modem | Power on write protection |
| LUN5 | Protected RW area | Modem FS1, modem FS2, FSC | Read/write |
| LUN6 | Customer | | Customer determined |
| LUN7 | Customer | | Customer determined |

**NOTE:** Cmnlib, lksecapp, and devinfo partitions are for the Android Verified Boot feature in LK; they store the devinfo structure in the user or RPMB partition.

**NOTE:** The PMIC partition is the PMIC configuration that is loaded and executed by XBL

**NOTE:** dpo (debug policy override), msadp (modem debug policy), and apdp (application debug policy) are used by the debug policy to enable debug on JTAG and SECURE BOOT enabled devices. Refer to *MSM8994/MSM8992/MSM8952 Debug Policy User Guide* [80-NU498-1] for more information.

# 4 GPT image generation

## 4.1 Generating GPT with ptools.py

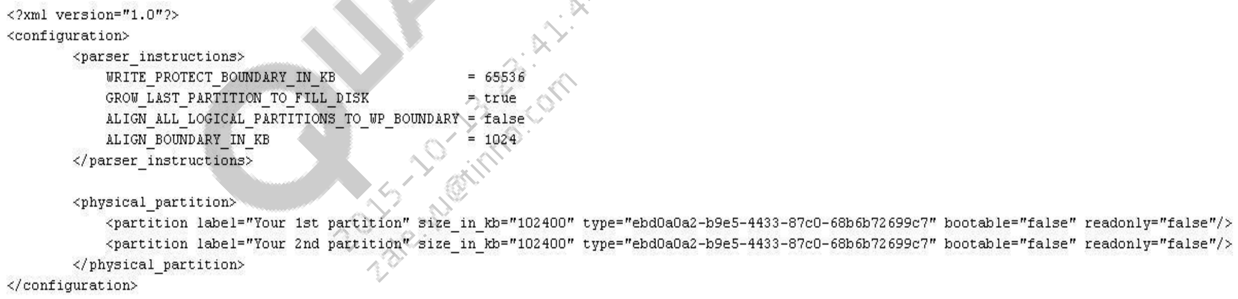
Run the following command exactly as shown:

```
python ptool.py -x partition.xml
```

**NOTE:** GPT is autodetected if GUIDs are present in partition.xml.

### 4.1.1 Input partition.xml

ptool.py works on partition.xml and calculates these files. Figure 3-1 shows the partition tables, including instructions on how to load to a device and how to patch.



```
<?xml version="1.0"?>
<configuration>
        <parser_instructions>
            WRITE_PROTECT_BOUNDARY_IN_KB                = 65536
            GROW_LAST_PARTITION_TO_FILL_DISK            = true
            ALIGN_ALL_LOGICAL_PARTITIONS_TO_WP_BOUNDARY = false
            ALIGN_BOUNDARY_IN_KB                        = 1024
        </parser_instructions>

        <physical_partition>
            <partition label="Your 1st partition" size_in_kb="102400" type="ebd0a0a2-b9e5-4433-87c0-68b6b72699c7" bootable="false" readonly="false"/>
            <partition label="Your 2nd partition" size_in_kb="102400" type="ebd0a0a2-b9e5-4433-87c0-68b6b72699c7" bootable="false" readonly="false"/>
        </physical_partition>
</configuration>
```

**Figure 3-1 Partition tables**

### 4.1.2 Output files

The following files are output:

- gpt_main0.bin – Primary GPT header and partition tables
- gpt_backup0.bin – Backup GPT header and partition tables
- rawprogram0.xml – How to program files to the device
- patch0.xml – How to patch zeros in the partition tables

The generated partition tables are meant to go onto any size card, for example, 1 GB, 2 GB, or 4 GB. Therefore, the partition table has zeros that must be patched later with the final partition size and card size.

## 4.2 Sparse image support

Android builds have supported generation of sparse ext4 images as part of the build system. Sparse ext4 images are now supported in Android bootloader for fastboot and generation of sparse images is enabled by default. Sparse ext4 images help address some size problems with the ex4 file system and affect generation of system.img.ext4, userdata.img.ext4, persist.img.ext4, cache.img.ext4, and tombstones.img.ext4.

T32 and QPST cannot uncompress the ext4 images. Therefore, a tool such as checksparse.py is required to regenerate the new rawprogram0.xml.

checkparse.py looks for the tag sparse="true" in rawprogram0.xml. The original partition.xml file must have marked the file as sparse. When ptool.py parses this file, this sparse tag is propagated to rawprogram0.xml.

```
partition.xml
<extended order="9" type="83" label="USERDATA" size="12165824"
readonly="false" sparse="true">
  <file name="userdata.img.ext4" offset="0"/>
</extended>
```

```
rawprogram0.xml
<program file_sector_offset="0" filename="userdata.img.ext4"
label="USERDATA" num_partition_sectors="12165824"
physical_partition_number="0" size_in_KB="5940.0" sparse="true"
start_byte_hex="0x84800" start_sector="1060"/>
```

Run checksparse.py, such as in the following example:

```
python checksparse.py -i rawprogram0.xml -s C:\path1 -s C:\path2 -s
C:\path3
```

To preserve your original rawprogram0.xml, you can specify an output filename, such as in the following example:

```
python checksparse.py -i rawprogram0.xml -o rawprogram0_unsparse.xml -s
C:\path1 -s C:\path2 -s C:\path3
```

checksparse.py requires the search paths (-s c:\path) to find the original sparse file at some path, and then break it into its smaller chunks.

## 4.3 parseGPT.pl

The parseGPT.pl tool can be used to verify the binary GPT image.

```
Usage: parseGPT.pl – decodes GPT partition tables
```

# 5 Programming tools

## 5.1 Programming procedures on A-Family chip

### 5.1.1 Flash programmer for A-Family chip

NOTE: This section was added to this document revision.

The emergency downloader is also known as the eMMC boot loader.

#### 5.1.1.1 Creating a HEX file (MPRG8960.hex)

The following example creates a HEX file:

```
build\ms>build emmcbld BUILD_ID=AAABQNLG
```

#### 5.1.1.2 Creating single-image MBN file (8960_msimage.mbn)

To create a single-image MBN file:

1. Go to the boot loader build folder **boot_images\core\storage\tools\jsdcc\partition_load_pt**.
2. Copy ptool.py, singleimage.py, and msp.py to a directory, for example, **d:\test**.
3. Copy sbl1.mbn, sbl2.mbn, sbl3.mbn, rpm.mbn, and tz.mbn to the same directory.
4. Type the command: **`python singleimage.py -x singleimage_partition_8960.xml`**.
5. The output file is called singleimage.bin. Rename it to 8960_msimage.mbn and save it to the location of your choice.

### 5.1.2 Programming with T32

#### 5.1.2.1 T32 environment

From the folder path **common\t32\t32_dynamic** in the meta build:

1. Double-click **Dynamic_ARM7_RPM_usb** to open the RPM T32 window.
2. Double-click **Dynamic_Krait_c0_usb** to open the apps T32 window.

#### 5.1.2.2 Erasing and programming all images

Type **do std_loadbuild.cmm fastboot** in the T32 RPM window.

This loads the RPM, boot loaders, and partition table using JTAG, reboots the system, and loads Android images and NON-HLOS.bin (peripheral images) using fastboot.

#### 5.1.2.3 Erasing and programming boot loaders

If the fastboot parameter is not provided, the script loads only the RPM image and boot loaders.

#### 5.1.2.4 Erasing only

To erase the chips, type **do std_loadbuild.cmm erase**.

### 5.1.3 Programming with QPST

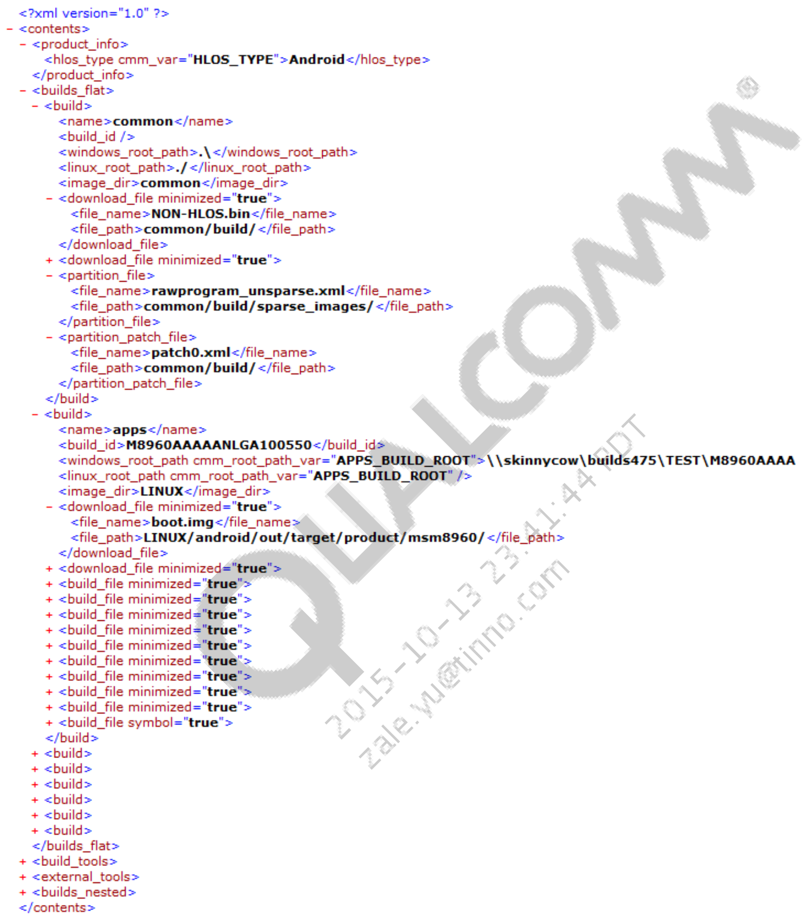
The main tool supported for programming is QPST. The following procedure requires version 2.7.375 or greater.

Licensees can use this document to make their own Flash programming tool if they do not want to use the tools provided by QTI. There might be updates to this specification from time to time to add new features.

### 5.1.3.1 Input contents.xml

Contents.xml is the input file for QPST eMMC software download tools. Contents.xml is located in the root of the metabuild directory.

Clicking **Load Build Contents** supplies the contents.xml file and eliminates the need to supply partition.xml, rawprogram0.xml, or patch0.xml.

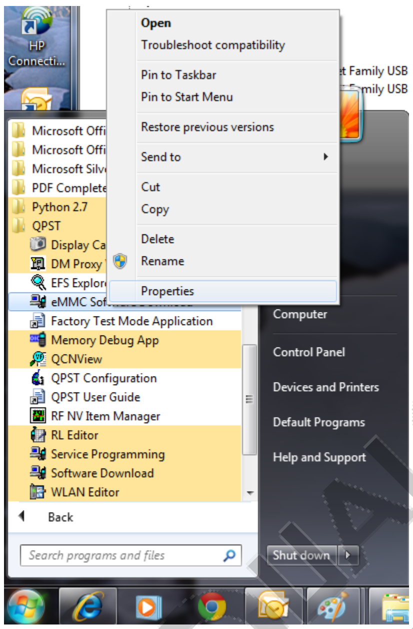


```
  <?xml version="1.0" ?>
- <contents>
  - <product_info>
      <hlos_type cmm_var="HLOS_TYPE">Android</hlos_type>
    </product_info>
  - <builds_flat>
    - <build>
        <name>common</name>
        <build_id />
        <windows_root_path>.\</windows_root_path>
        <linux_root_path>./</linux_root_path>
        <image_dir>common</image_dir>
      - <download_file minimized="true">
          <file_name>NON-HLOS.bin</file_name>
          <file_path>common/build/</file_path>
        </download_file>
      + <download_file minimized="true">
      - <partition_file>
          <file_name>rawprogram_unsparse.xml</file_name>
          <file_path>common/build/sparse_images/</file_path>
        </partition_file>
      - <partition_patch_file>
          <file_name>patch0.xml</file_name>
          <file_path>common/build/</file_path>
        </partition_patch_file>
      </build>
    - <build>
        <name>apps</name>
        <build_id>M8960AAAAANLGA100550</build_id>
        <windows_root_path cmm_root_path_var="APPS_BUILD_ROOT">\\skinnycow\builds475\TEST\M8960AAAA
        <linux_root_path cmm_root_path_var="APPS_BUILD_ROOT" />
        <image_dir>LINUX</image_dir>
      - <download_file minimized="true">
          <file_name>boot.img</file_name>
          <file_path>LINUX/android/out/target/product/msm8960/</file_path>
        </download_file>
      + <download_file minimized="true">
      + <build_file minimized="true">
      + <build_file minimized="true">
      + <build_file minimized="true">
      + <build_file minimized="true">
      + <build_file minimized="true">
      + <build_file minimized="true">
      + <build_file minimized="true">
      + <build_file minimized="true">
      + <build_file minimized="true">
      + <build_file symbol="true">
      </build>
    + <build>
    + <build>
    + <build>
    + <build>
    + <build>
    + <build>
    </builds_flat>
  + <build_tools>
  + <external_tools>
  + <builds_nested>
  </contents>
```

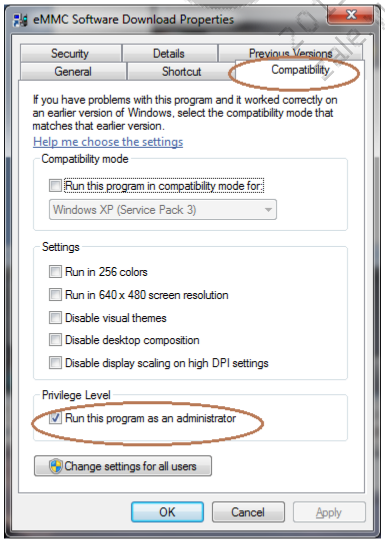
### 5.1.3.2 Running as administrator

For eMMC Software Download to program the images, the tools must be run as an administrator. The following procedure applies to Windows® 7.

1. From the Start menu, right-click **eMMC Software Download** and select **Properties**.



2. Select the **Compatibility** tab and check **Run this program as an administrator**.


eMMC Software Download Properties

Security | Details | Previous Versions
General | Shortcut | Compatibility

If you have problems with this program and it worked correctly on an earlier version of Windows, select the compatibility mode that matches that earlier version.

Help me choose the settings

Compatibility mode

☐ Run this program in compatibility mode for:

Windows XP (Service Pack 3)

Settings

☐ Run in 256 colors

☐ Run in 640 x 480 screen resolution

☐ Disable visual themes

☐ Disable desktop composition

☐ Disable display scaling on high DPI settings

Privilege Level

☑ Run this program as an administrator

Change settings for all users

OK | Cancel | Apply

### 5.1.3.3 Programming in two phases

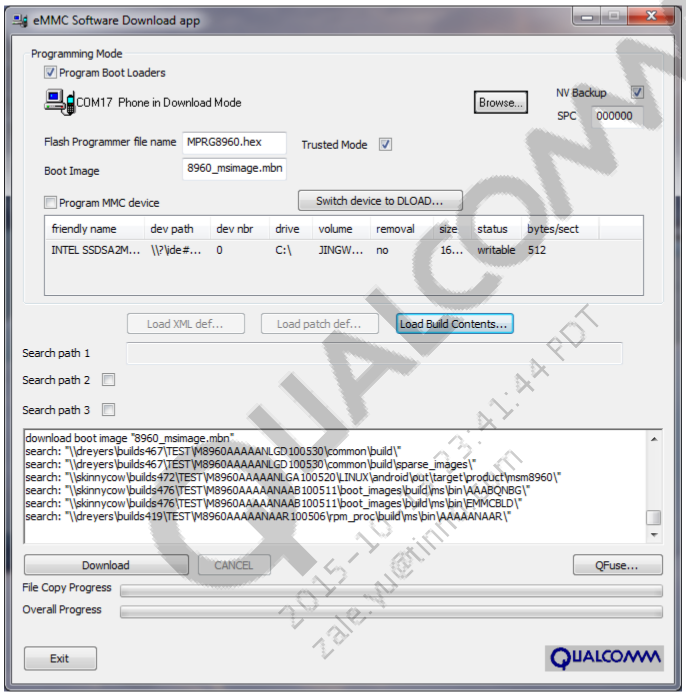
In some cases the device does not reboot by itself after programming the boot loaders. Therefore, the programming is performed in two phases. (This is a known issue. A future release of QPST or device firmware might fix this issue.)

#### 5.1.3.3.1 Phase 1 – Programming boot loaders

To program the boot loaders:

3. Check **Program Boot Loaders** and uncheck **Program MMC device**.
4. Click **Load Build Contents** and provide the contents.xml file.
5. Click **Download**.

   The programming of boot loaders occurs. This process lasts for a few minutes. At the end, if a warning about timeout appears, ignore it.



| friendly name | dev path | dev nbr | drive | volume | removal | size | status | bytes/sect |
|---|---|---|---|---|---|---|---|---|
| INTEL SSDSA2M... | \\?\ide#... | 0 | C:\ | JINGW... | no | 16... | writable | 512 |

```
download boot image "8960_msimage.mbn"
search: "\\dreyers\builds467\TEST\M8960AAAAANLGD100530\common\build\"
search: "\\dreyers\builds467\TEST\M8960AAAAANLGD100530\common\build\sparse_images\"
search: "\\skinnycow\builds472\TEST\M8960AAAAANLGA100520\LINUX\android\out\target\product\msm8960\"
search: "\\skinnycow\builds476\TEST\M8960AAAAANAAB100511\boot_images\build\ms\bin\AAABQNBG\"
search: "\\skinnycow\builds476\TEST\M8960AAAAANAAB100511\boot_images\build\ms\bin\EMMCBLD\"
search: "\\dreyers\builds419\TEST\M8960AAAAANAAR100506\rpm_proc\build\ms\bin\AAAAANAAR\"
```

6. Reboot the device.

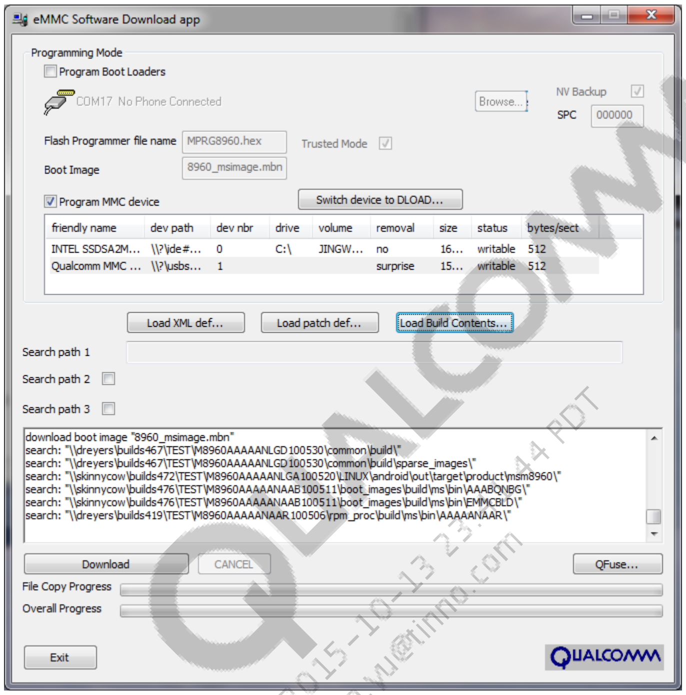
   Remove the battery or power supply and the USB cable to force a power cycle. The device should reboot and enumerate as a mass storage device.

**NOTE:** If watchdog is enabled, use emmcbld to send a reset command to reset the device.

#### 5.1.3.3.2 Phase 2 – Programming an MMC device

To program an MMC device:

1. Uncheck Program Boot Loaders and check Program MMC device.
2. Click **Load Build Contents** and provide the contents.xml file.
3. Click **Download**.



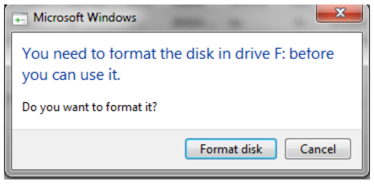
SPC: 000000
Flash Programmer file name: MPRG8960.hex
Trusted Mode: ✓
Boot Image: 8960_msimage.mbn
Program MMC device: ✓

| friendly name | dev path | dev nbr | drive | volume | removal | size | status | bytes/sect |
|---|---|---|---|---|---|---|---|---|
| INTEL SSDSA2M... | \\?\ide#... | 0 | C:\ | JINGW... | no | 16... | writable | 512 |
| Qualcomm MMC ... | \\?\usbs... | 1 | | | surprise | 15... | writable | 512 |

Search path 1
Search path 2
Search path 3

```
download boot image "8960_msimage.mbn"
search: "\\dreyers\builds467\TEST\M8960AAAAANLGD100530\common\build\"
search: "\\dreyers\builds467\TEST\M8960AAAAANLGD100530\common\build\sparse_images\"
search: "\\skinnycow\builds472\TEST\M8960AAAAANLGA100520\LINUX\android\out\target\product\msm8960\"
search: "\\skinnycow\builds476\TEST\M8960AAAAANAAB100511\boot_images\build\ms\bin\AAABQNBG\"
search: "\\skinnycow\builds476\TEST\M8960AAAAANAAB100511\boot_images\build\ms\bin\EMMCBLD\"
search: "\\dreyers\builds419\TEST\M8960AAAAANAAR100506\rpm_proc\build\ms\bin\AAAAANAAR\"
```

A few minutes are required to finish this phase of programming. If Windows displays a warning about formatting the disk, click **Cancel** to ignore the warning.



You need to format the disk in drive F: before you can use it.

Do you want to format it?

Wait for the programming to finish and reboot the device again. The sparse images programming is complete and the device boots up and is ready to use.

## 5.2 Programming procedures on B-Family chip

This section details the programming procedures on a B-Family chip, for example, AMSS8974.

### 5.2.1 Flash programmer for B-family chip

NOTE: This section was added to this document revision.

B-Family chipsets use firehose protocol in flash programmer to download images. To create firehose programmer, use the following command (8994 for example):

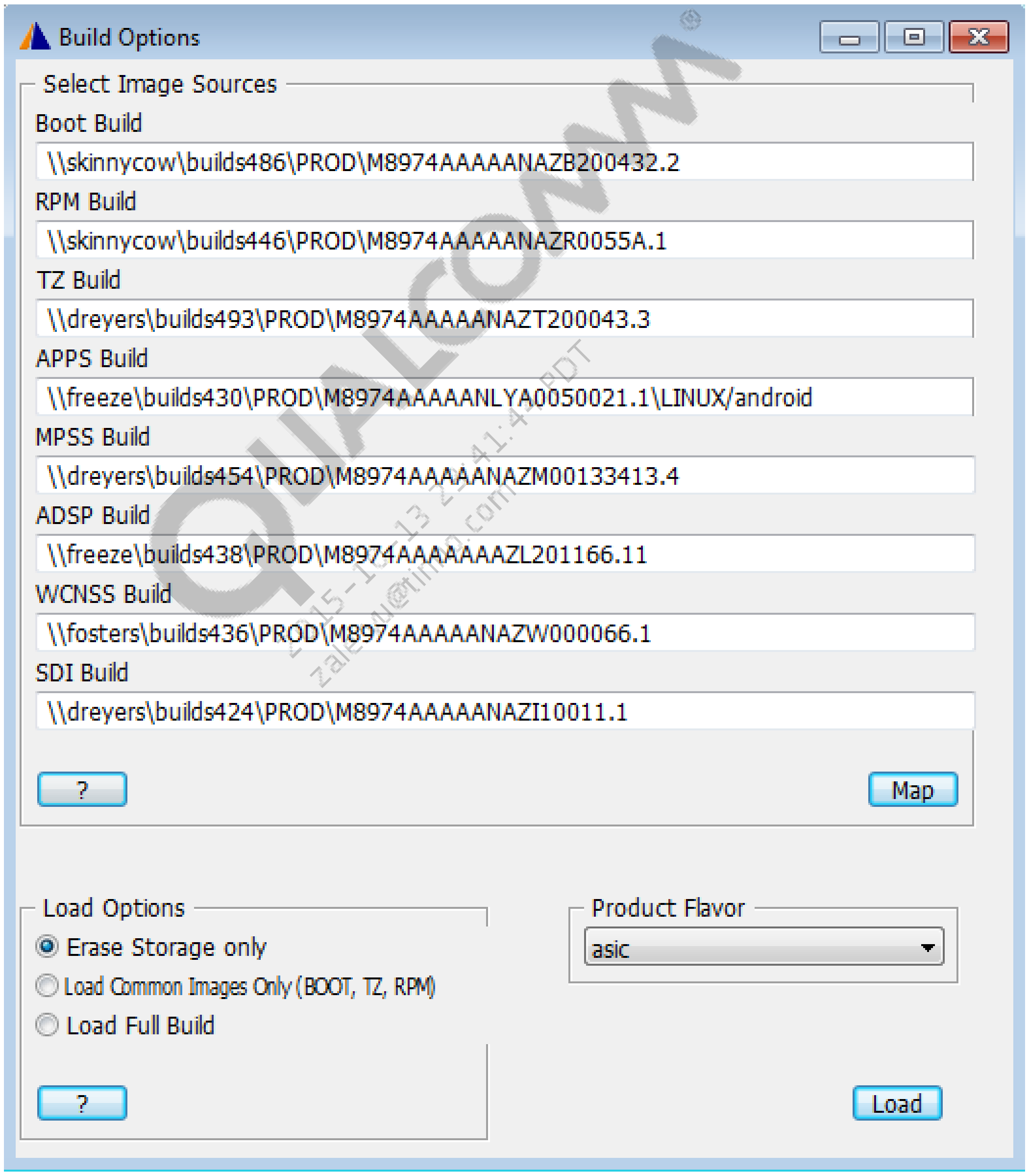
```
build\ms>b8994 deviceprogrammer
```

Two files are listed in the path **boot_images\build\ms\bin\8994**:

- prog_emmc_firehose_8994_ddr.mbn --- The flash programmer that runs in DDR
- prog_emmc_firehose_8994_lite.mbn --- The flash programmer that runs in IMEM

### 5.2.2 Erasing eMMC with TRACE32

1. Select **RPM** and click **Start**.
2. Select **APPS_CORE0** and click **Start**.
3. In the RPM TRACE32® window, select **RPM commands** > **Build Options,** then select the ASIC type.
4. Click **Load** in the Build Options window.

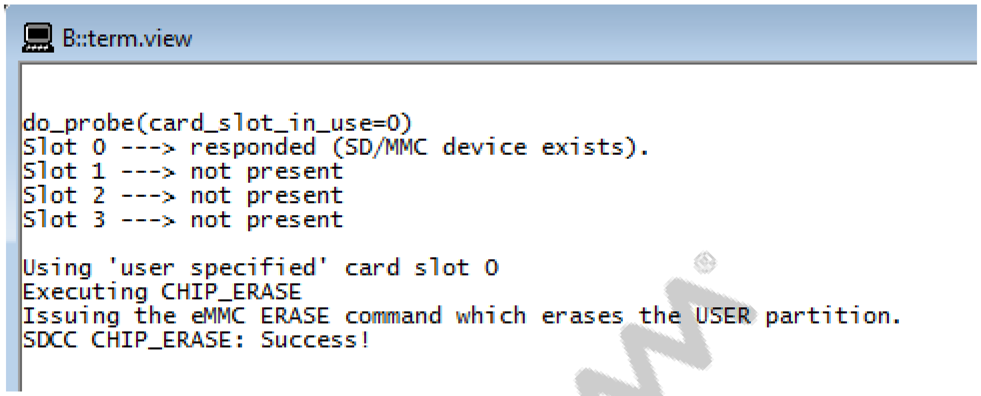


Boot Build
\\skinnycow\builds486\PROD\M8974AAAAANAZB200432.2
RPM Build
\\skinnycow\builds446\PROD\M8974AAAAANAZR0055A.1
TZ Build
\\dreyers\builds493\PROD\M8974AAAAANAZT200043.3
APPS Build
\\freeze\builds430\PROD\M8974AAAAANLYA0050021.1\LINUX/android
MPSS Build
\\dreyers\builds454\PROD\M8974AAAAANAZM00133413.4
ADSP Build
\\freeze\builds438\PROD\M8974AAAAAAAZL201166.11
WCNSS Build
\\fosters\builds436\PROD\M8974AAAAANAZW000066.1
SDI Build
\\dreyers\builds424\PROD\M8974AAAAANAZI10011.1

[✓ Erase Storage only] Load Common Images Only (BOOT, TZ, RPM) Load Full Build

Product Flavor: asic

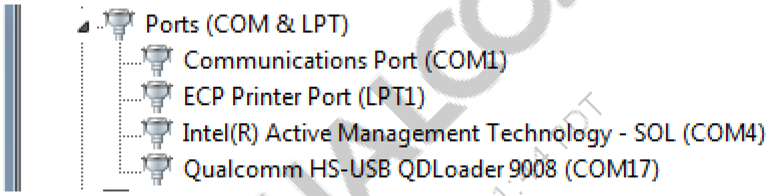
5. In the APPS_CORE0 T32 window, a success status is printed.



```
do_probe(card_slot_in_use=0)
Slot 0 ---> responded (SD/MMC device exists).
Slot 1 ---> not present
Slot 2 ---> not present
Slot 3 ---> not present

Using 'user specified' card slot 0
Executing CHIP_ERASE
Issuing the eMMC ERASE command which erases the USER partition.
SDCC CHIP_ERASE: Success!
```

The device enumerates as emergency download mode 9008 in the device manager.



```
Ports (COM & LPT)
    Communications Port (COM1)
    ECP Printer Port (LPT1)
    Intel(R) Active Management Technology - SOL (COM4)
    Qualcomm HS-USB QDLoader 9008 (COM17)
```

## 5.2.3 Programming with QPST

QFIL is now supported for all B-Family chipsets image programming. Refer to [80-NN120-1] *Qualcomm Flash Image Loader (QFIL) User Guide* for details.

# A References

## A.1 Related documents

| Title | Number |
|---|---|
| **Qualcomm Technologies, Inc.** | |
| *Application Note: Embedded Multimedia Card (eMMC) Software Download Tool* | 80-N2967-1 |
| *Application Note: eMMC Partition Tables and Raw Programming* | 80-N4584-1 |
| *Qualcomm Flash Image Loader (QFIL) User Guide* | 80-NN120-1 |
| *MSM8994/MSM8992/MSM8952 Debug Policy* | 80-NU498-1 |
| **Standards** | |
| *Universal Flasg Storage (UFS) Unified Memory Extention* | JESD220-1 |
| **Resources** | |
| *GUID partition table from Wikipedia* | http://en.wikipedia.org/wiki/GUID_Partition_Table |

## A.2 Acronyms and terms

| Acronym or term | Definition |
|---|---|
| GPT | Globally Unique Identifier Partition Table |
| GUID | Globally Unique Identifier |
| LBA | Logical Block Addressing |
| QPST | Qualcomm Product Support Tool |