Firmware

From wikiPodLinux

This page describes how executable code on the iPod is stored and executed.

Table of contents

Startup Process

When power is first applied to the iPod the hardware starts execution of code located at the ARM boot location which is 0x0. This location normally maps to the flash. The first flash sector contains the ARM reset vectors with the reset vector pointing to the location where the **bootloader** is stored. The bootloader then executes and in the simplest case loads operating system from the HDD and then executes that.

When ipodlinux is installed some additional code is appended to the operating system and the start location is set to this new code. The ipodlinux bootloader code simply looks for a key press then copies either the Linux or Apple code to the correct location and then executes that.

Firmware updates

When Apple provides new versions of its software of the iPod their updater program contains **firmware** images which contain all the code that is executed on the iPod. The Windows updater stores these images as resources within the executable and the Mac version as files in the Updates directory in the updater package.

Below are the Windows resource names and the iPod models they correspond to. (Versions from iPod Updater 2006-01-10)

25.6.2

Windows Firmware Resources

Resource Name	iPod model
IDR_FIRMWARE-1.1.5	iPod 1/2g (with touch wheel or scroll wheel)
IDR_FIRMWARE-2.2.3	iPod 3g (with dock connector)
IDR_FIRMWARE-4.3.1.1	iPod 4g (with click wheel)
IDR_FIRMWARE-10.3.1.1	HP iPod 4g [same as IDR_FIRMWARE-4.3.1.1]
IDR_FIRMWARE-5.1.2.1	iPod photo (with color display)

IDR_FIRMWARE-11.1.2.1	HP iPod photo [same as IDR_FIRMWARE-5.1.2.1]
IDR_FIRMWARE-13.1.2.1	iPod 5g (Video)
IDR_FIRMWARE-20.1.2.1	iPod 5.5 (5g enhanced)
IDR_FIRMWARE-25.1.2.1	iPod 5.5g (5g enhanced)
IDR_FIRMWARE-24.1.0	iPod classic
IDR_FIRMWARE-3.1.4.1	iPod mini
IDR_FIRMWARE-6.1.4.1	HP iPod mini [same as IDR_FIRMWARE-3.1.4.1]
IDR_FIRMWARE-7.1.4.1	iPod mini 2g
IDR_FIRMWARE-14.1.3.1	iPod nano 1st gen
IDR_FIRMWARE-17.1.3.1	iPod nano 1st gen
IDR_FIRMWARE-19.1.1.3	iPod nano 2nd gen
IDR_FIRMWARE-29.1.1.3	iPod nano 2nd gen
IDR_FIRMWARE-26.1.0	iPod nano 3rd gen
IDR_FIRMWARE-128.1.1.5	iPod Shuffle 1st gen
IDR_FIRMWARE-129.1.1.5	iPod Shuffle 1st gen
IDR_FIRMWARE-130.1.0.3	iPod Shuffle 2nd gen
IDR_FIRMWARE-131.1.0.3	iPod Shuffle 2nd gen

Each of these files are an exact copy of what goes into the firmware partition on an iPod disk.

Firmware Partition format

The firmware partition is the first partition on an iPod (after that, usually either a HFS+ or a FAT32 partition follows, while iPodLinux may add also a ext2 partition). This partition **usually starts at logical block 63**, on both MacPods and WinPods, with blocks being 2048 bytes in size on 5.5G iPods, and 512 bytes on all previous models. This means that, when dealing only with 512 byte-sized blocks, the new 5.5G iPods have their firmware partition start at block 252 instead of 63.

> **Note: From here on, blocks *always* mean 512 byte sectors**

The firmware partition contains a very simplified file system. The basic layout is as follows:

- block offset 0: Volume Header
- block offset 1: Start of Volume Space (offsets in the Directory and Volume Header may be relative to this point, depending on the *firmware format version*)
- block offset 33: Directory block ("bootloader table")
- block offset 34: Start of file area (with *firmware format* 3, the first file actually starts at block 35, though)

To view this partition and its contents conveniently, rohPod can be used.

Volume Header

The first 256 bytes of this block contain a distinctive *S T O P* sign text.

Starting at 0x100 is the following structure.

iPod Firmware Header

offset	length	description
0x100	4 bytes	magic value "[hi]"
0x104	4 bytes	location of the Directory as a byte offset from partition start (always 0x4000)
0x108	2 bytes	location of the extended header (0x10c)
0x10a	2 bytes	**firmware format version**

The value at 0x10a is either a '2' for 1st, 2nd and 3rd generation iPods or a '3' for mini, photo, 4th and 5th generation iPods. Only very old 1G models that were never updated may still carry a 1 here.

Directory

Starting at the location specified by 0x104 is the Directory (*boot image structure list*), which is usually at block 33 (offset 0x4200) in the partition: The directory offset is 0x4000 (32 blocks), but since the the dir is part of the *Volume Space', which starts at block 1, the final offset is 0x4200.*

> (Note: Version 1 of the firmware did not use the *Volume Space* at all. Instead, it placed the directory at block 32 (offset 0x4000 of the firmware partition (block 95 on the disk), and used disk-wide offsets for the file locations in the directory. E.g, the "osos" file, which usually starts at offset 0x4400, would have the *entryoffset* 0xC200 (which is 0x4400 + 0x7F00 for the partition start) in the dir at offset 0x4000.) That's why you might actually find two similar looking directories inside a Firmware file or partition - the older one at block 95 from start of the disk is only relevant if it is installed on an iPod that still has old bootstrap code using the old addressing scheme. Once you install a current Firmware on it, however, the iPod will install the new Flash ROM which contains bootstrap code that uses only the new scheme.

The directory is very simple in its structure: It contains a list of entries of 40 bytes each, and it does not know subdirectories, only one flat root dir. It is not known how many entries may be used at maximum, but iPodLinux tools currently support up to 10 entries.

Directory Entry structure

Bootloader Image

offset	length	description
dev	32bit	"ATA!", or 0x00000000 if entry is unused
type	32bit	Fle name ("osos" for Apple's main software, "aupd" for Flash ROM update)
id	32bit	Usually zero (set to 1 for "aupd" once flash-ROM update has been performed)
devOffset	32bit	Offset in bytes for start of file's data (see *Note 1* below)
len	32bit	Length in bytes of file (a file occupies always occupies all blocks in sequence, there's no fragmentation possible)
addr	32bit	Load address, the file will be loaded to this address in memory
entryOffset	32bit	Execution start within file
chksum	32bit	Checksum for file
vers	32bit	File version
loadAddr	32bit	Load address for file (normally 0xffffffff except during Flash updates)

These values are all in little endian (Intel) order.

Note 1: Each *firmware format version* so far used a different way to locate the start file block:

- Version 1 used disk-relative offsets, i.e. starting with the first block of the disk. E.g, the "osos" file would have a value of 0xC200 in the dir, meaning that the file starts at block 97 from start of the disk.
- Version 2 uses partition-relative addressing. E.g. the "osos" file would have a value of 0x4400 in the dir, meaning that the file starts at block 34 from start of the partition (and with the partition starting at blk 63, the file would start at block 97).
- Version 3 uses *Volume Space*-relative addressing. As shown above, that starts at block 1 of the partition. E.g. the "osos" file would have a value of 0x4400 in the dir, meaning that the file starts at block 34 from start of the Volume Space, which is block 35 in the partition and block 98 on the disk.

Note 2: In addtion, for firmware version 3 files of type "aupd" the actual image data is encoded using some unknown format. If you want to extract the unencoded Flash ROM contents, you can use the *getflash* tool which can be found in the Subversion repository instead.

You can find code that calculates the checksum and other values in CVS in the *bootloader* and the *patch_fw* modules.

Sub-Image entries

The iPodLinux bootloaders support so-called **sub-images**, which are a further segmentation of a file in the main directory. Note that this format is specific to iPL and not supported by Apple.

A file with sub-images is identified by the value 1 (instead of 0) in the byte at offset 11 in a directory entry if the directory was written with make_fw2 (which is also used by Installer 2). If the older make_fw is used, such a marker is currently not set, meaning that one needs to read the actual block at *devOffset* to see if subimgs are present in the file.

The sub-img information is an array of directory entries just like the ones for the main directory. The location is at offset 0x100 from the file's entry point. Up 5 entries can be found there. From the view of make_fw and the bootloaders these entries are numbered 0 to 4. Each of these entries, when used, has the same ID as their master entry in the root directory. There's also a virtual entry marked with @ (e.g. "osos@"), which is used by make_fw to refer to the part of the file that starts at the execution offset and ends with the end of the entire file. This is usally the bootloader code.

Note: The offsets inside sub-img entries are always relative to the start of the partition and not relative to the *Volume Space*, i.e. they ignore the extra block offset used by Apple in new directory versions.

rohPod (version 1.0.5 and later) can show these sub-img entries rather comfortably by viewing the firmware partition of an iPod.

Extracting firmwares under Windows

If you want to extract firmware files from the ipod updater you can do it easily.

Download apple ipod updater (*https://www.apple.com/ipod/download/*) and install it.

Download resource hacker (*http://www.users.on.net/johnson/resourcehacker/*) and open the "iPod Updater xxxx-xx-xx.exe" with the resource hacker program (you will find it in the folder where you installed ipod-updater).

You will be able to see all firmwares that are stored in 'ipod updater.exe' and you can extract it clicking in the firmware that you want to extract and then clicking in the 'actions' menu and in the 'save resource as binary file' option.

Extracting the OS Code

To extract the operating system code (known internally as the **retailos**) in order to disassemble we just need to extract the right portion of the firmware update file. Essentially the data from the **devOffset** for **len** bytes. Using make_fw under Unix or Linux, the following command would extract the retailos to a file retailos.bin from the 2.2.2 firmware image (3rd gen).

```
dd if=FIRMWARE-2.2.2 of=retailos.bin bs=1 skip=17408 count=3276232
```

The easier way, however, is to use the make_fw tool for this task:

```
./make_fw -o retailos.bin -e 0 FIRMWARE-2.2.2
```

To disassemble this with the GNU tools from the toolchain:

```
arm-elf-objdump --target=binary --architecture=arm --disassemble-all -z retailos.bin> retailos.bin
```

The disassembly will be a bit crazy as the file is in binary mode so it is all treated as code (normally the data portions look like invalid assembly).

Here is some quick-and-dirty perl code to extract the file sections. This has been tested on Windows, it seems to work (note: it's probably easier to use make_fw):

```
$filename = pop;
open (FH, $filename) || die "Can't open file: $filename\n";
binmode (FH);

$address = 0x100;
seek(FH, $address, 0) or die "Can't seek to address: $address\n";
read(FH, $magic, 4);
$magic = reverse $magic;
#print $magic;
if ($magic ne "[hi]")
{
  die "Can't find magic string at $address\n";
}
else
{
  print "Found magic string: $magic at address: $address\n";
}

$address = 0x104;
seek(FH, $address, 0) or die "Can't seek to address: $address\n";
read(FH, $dword, 4);
$section_table_loc = unpack ("l", $dword);
#print $section_table_loc;
print "Section table offset is: $section_table_loc\n";

for (0..11)
{
  $address = $section_table_loc + 512 + (40 * $_);
  seek(FH, $address, 0) or die "Can't seek to address: $address\n";

  read (FH, $record, 40);
  ($dev, $type, $id, $devOffset, $len, $addr, $entryOffset, $chksum, $vers, $loadAddr) =
    unpack ("A4 A4 l l l l l l l l", $record);
  $dev = reverse $dev;
  $type = reverse $type;
  if ($type ne "")
  {
#    printf ("$dev $type 0x%x 0x%x 0x%x\n", $devOffset, $len, $addr);
    if ($type eq "osos" || $type eq "aupd")
    {
      $count{$type}++;
      $outfilename = $filename.".".$type.$count{$type};
      printf ("Attempting to extract section $outfilename at offset 0x%x\n", $devOffset);
      $address = $devOffset;
      seek(FH, $address, 0) or die "Can't seek to address: $address\n";
      if (read (FH, $blob, $len) == $len)
      {
        open (OF, "> $outfilename");
        binmode (OF);
        print OF $blob;
        close (OF);
        printf ("Success. $len bytes written, memory address is 0x%x\n", $addr);
      }
      else
      {
        die "Could not read all of contents for section: $outfilename\n";
      }
    }
  }
}

close (FH);
```

Here's a sample run:

```
D:\disassemble\ipod>perl extract.pl IDR_FIRMWARE-2.2.2
Found magic string: [hi] at address: 256
Section table offset is: 16384
Attempting to extract section IDR_FIRMWARE-2.2.2.osos1 at offset 0x4400
Success. 3276232 bytes written, memory address is 0x28000000
Attempting to extract section IDR_FIRMWARE-2.2.2.aupd1 at offset 0x324200
Success. 1113000 bytes written, memory address is 0x28000000
```

Inside the flash update

More interesting is the contents of the flash update image (the dpua or so called **softupd** image). This code is executed to flash your iPod with a new firmware which updates the bootloader (as well as the diagnostic and disk mode code).

If you extract the entire flashupd image and diassemble it you can locate the individual images that are the boot loader, diagnostic, disk image, and disk scan code.

For the 2.2.2 firmware though here are the commands.

```
dd if=FIRMWARE-2.2.2 of=bootloader.bin bs=1 skip=3374476 count=69392
dd if=FIRMWARE-2.2.2 of=diag.bin bs=1 skip=4118848 count=84192
dd if=FIRMWARE-2.2.2 of=diskscan.bin bs=1 skip=4203040 count=97628
dd if=FIRMWARE-2.2.2 of=diskmode.bin bs=1 skip=4266172
```

Unfortunately the iPod mini and later models have their flashupd code encoded but BadBlox found out how to decrypt it. For more information, visit Flash Decryption.

Reverse Engineering

As noted above the GNU tools can provide some basic disassembly functions, the commercial tool IDA Pro (*http://www.datarescue.com/idabase/*) is recommended for serious work.

Some results from the reverse engineering can be found on the PP5002 and PP5020 pages.

Retrieved from "http://ipodlinux.org/Firmware"

Categories: Hardware

- This page was last modified 22:26, 6 Sep 2017.
-