The Uncoöperative Organization

Programming and other human stuff.

- RSS

Search

Navigate… ▼

- Blog
- Archives

The EFI System Partition and the Default Boot Behavior

Feb 6th, 2014

A lot of the time, we talk about creating a partition to serve as the *EFI System Partition*. This partition is mandated by the UEFI specification for several tasks. Adam covered what's going on at a relatively high level on his recent blog post, and you should read the whole thing:

> An ‘EFI system partition’ is really just any partition formatted with one of the UEFI spec-defined variants of FAT and given a specific GPT partition type to help the firmware find it. And the purpose of this is just as described above: allow everyone to rely on the fact that the firmware layer will definitely be able to read data from a pretty ‘normal’ disk partition.
>
> **Adam Williamson** *www.happyassassin.net/2014/01/...*

There’s nothing truly special about an ESP. It isn’t an ESP because of the GPT GUID and label, nor because of the file system type. Those are how the firmware identifies a partition, and the file system it contains, as candidates to treat as the ESP, when it really needs to find one. The only factor in determining if a partition is the ESP is this: is the firmware attempting to use it as the ESP?

At the same time, the requirements for the ESP give us latitude; we know that we can use UEFI’s APIs to find correctly constructed FAT file systems, but there’s no need for those to be the ESP. In fact, even when we create multiple partitions with the ESP’s GUID and label, there’s no requirement that the firmware looks at more than one of them if it needs to find the ESP, and there’s no guarantee as to which one it will pick, either.

Normal booting

Thankfully, most of the time we don't care. Under normal circumstances, the system isn't actually looking for something to fill the role of the ESP. When the system starts up, it consults the `BootOrder`, which is just a list of 16-bit numbers, such as `BootOrder: 0001,001A,0003`. When this is found, the firmware will iterate the list, as you might expect. For each entry in the list, it looks for a corresponding `Boot####` variable — `Boot0001` for 0001, `Boot001A` for 0x001a, and so on, with the value from `BootOrder` rendered in capitalized hexadecimal. If it doesn't find the variable, it continues to the next entry in `BootOrder`. If it does find the variable, it reads the contents of the variable to try to decide what to do. Generally speaking, what's in there is a human friendly label, a *Device Path* like `ACPI(a0341d0,0)PCI(1f,2)SATA(0,0,0)HD(1,800,64000,12029cda-8961-470d-82ba-aeb17dba91a5)File(\EFI\fedora\shim.efi)`, and optional data to be passed to whatever program is to be loaded. This Device Path reflects the series of parts that need to be initialized in order to find the ultimate media device `HD(1,800,64000,12029cda-8961-470d-82ba-aeb17dba91a5)`[1], which specifies a partition number, the partition's offset and size, and the partition's GUID.

This partition need not be an "EFI System Partition" properly; the firmware doesn't care if the GUID in the partition table is the ESP GUID, only that it matches the GUID in the Device Path. Likewise, it doesn't care about the label. This file system *may* be the ESP, but there's no requirement. The only real requirement here is that we have to use FAT, because it's the only file system the firmware is guaranteed to know about.

At this point, the system firmware is going to initialize each device that's part of that Device Path, in order, that hasn't already been initialized. Some things will always be initialized —

it's unlikely that ACPI tables and the PCI root hub specified in this particular path[2] weren't already needed to get this far — but other things, like the SATA controller or its port 0, port multiplier 0, LUN 0 device, `SATA(0,0,0)`[3], will probably need to be initialized. Once that is done, the firmware will examine the disk and see if it has a partition matching the `HD()` path above. If it finds that, and it contains a FAT file system, it looks for `File(\EFI\fedora\shim.efi)`.

If any of that goes wrong, it moves on to the next `Boot####` entry, and follow the exact same process, initializing one peripheral at a time until it can load a file from the FAT it finds.

What if everything we know is wrong?

If all of that fails, and there is no working boot entry found by traversing `BootOrder`, there are still things to do. At this point, the firmware will start initializing all peripherals it can find, in whatever order it happens to choose — for some peripherals this will be linear and predictable, for some it will start things all at once and they'll respond in whatever order they become ready. For each media device it finds, it will check if it's a removable device, such as optical media, or a fixed device like a hard drive. At this phase, the firmware only considers removable devices. On each of those, it looks for an `EFI System Partition`[4] with a FAT file system using whatever method it knows for that type of media, and if it finds one, it checks for`\EFI\BOOT\BOOTX64.EFI`[5].

Typically, if it finds a removable device it can boot, that's because you have your OS install image connected in some way, which may well be an indicator that you're trying to start the OS installer. It's also possible you've installed to removable media, or you're running off a live image.

If it doesn't find suitable removable media, or if a failure condition is returned by the application it tries to start, it continues traversing all removable media until it has exhausted the possibilities.

If it still hasn't found any removable media it can boot, the firmware will move on to fixed media. At this point, all media is probably discovered — you may have hot-plugged a USB stick or something, but don't plan on that sort of action working reliably at this stage. Since there's no real device or media discovery left to do, the firmware is going to iterate back over the devices once more, and will try to start each fixed media device in the same manner it previously did with the removable devices – it will look for an ESP, and try to start `\EFI\BOOT\BOOTX64.efi`. This is known as the "Default Boot Behavior".

The Default Boot Behavior

On Fedora systems, and likely any other OS using shim, we handle the Default Boot Behavior through a utility shim provides named `fallback.efi`. Our signed shim package provides several files, all of which reside on whichever partition we've mounted as the ESP, which is mounted at `/boot/efi`:

```
/boot/efi/EFI/BOOT/BOOTX64.EFI
/boot/efi/EFI/BOOT/fallback.efi
/boot/efi/EFI/fedora/BOOT.CSV
/boot/efi/EFI/fedora/MokManager.efi
/boot/efi/EFI/fedora/shim-fedora.efi
/boot/efi/EFI/fedora/shim.efi
```

When we're operating under the Default Boot Behavior, the system firmware launches the Device Path `File(EFI\BOOT\BOOTX64.EFI)`, which corresponds to `/boot/efi/EFI/BOOT/BOOTX64.EFI` on our file system. This is actually just another copy of `/boot/efi/EFI/fedora/shim.efi`, but it behaves slightly differently.

Upon startup, `shim` checks to see if it's been launched from `\EFI\BOOT`[6]. If it has, it checks to see if there's another file in this directory called `fallback.efi`, which our `shim` package provides on an installed system, but which we purposefully omit from removable media. If it finds `fallback.efi`, it executes it as a normal UEFI application.

fallback.efi

> It is expected that this default boot will load an operating system or a maintenance utility. If this is an operating system setup program it is then responsible for setting the requisite environment variables for subsequent boots.

UEFI Specification section 3.3 *Revision 2.4*

As you may well have come to realize, the purpose of `fallback.efi` is to rebuild boot options in the case that `BootOrder` or the `Boot####` variables it references have been destroyed, or the case that you've moved a fixed disk between machines with the intent on booting from it.

When fallback runs, it queries the firmware for the disk from which it was loaded. It then iterates over every subdirectory of `\EFI` on that disk that isn't `BOOT`, looking for files named `BOOT.CSV`. On Fedora, we provide such a file in `\EFI\fedora\BOOT.CSV`. This file is a UCS-2 file of comma separated values, and its contents look like this:

```
shim.efi,Fedora,,This is the boot entry for Fedora
```

For each valid entry in each BOOT.CSV file it finds, fallback creates a new `Boot####` variable and appends it to `BootOrder`. In this case it creates a boot entry with the label 'Fedora', with a Device Path that points to the disk fallback was run from, and the file path corresponding to the directory in which shim found this particular `BOOT.CSV`, with the first entry in the CSV appended: `File(\EFI\fedora\shim.efi)`

Once fallback has finished iterating all the CSV files in all the directories other than BOOT, it boots the first option it has added. On the next reboot, there will be a `BootOrder` variable, and its first entries will be whichever `Boot####` variables fallback created.

1. Astute readers will no doubt notice that often we have device paths with only the `HD(1,800,64000,12029cda-8961-470d-82ba-aeb17dba91a5)` and `File(\EFI\fedora\shim.efi)` components. The spec allows this explicitly, and in this case it initializes every peripheral in no particular order until it finds a partition matching that Hard Disk Media Device Path, and the appropriate file on it.↩

2. In this case `ACPI(a0341d0,0)` represents the PCI Express Root Port on my CPU, and `PCI(1f,2)` is what `lspci` whould show as `00:1f.2 SATA controller: Intel Corporation 6 Series/C200 Series Chipset Family SATA AHCI Controller (rev 05)`.↩

3. Yes, seriously, `SATA(0,0,0)` represents the disk and `HD(1,800,64000,12029cda-8961-470d-82ba-aeb17dba91a5)` represents the partition on it. I'm so sorry.↩

4. Strictly speaking, the firmware is required to check *one* partition which has the correct GUID and label in GPT and is a FAT file system. It is allowed to check partitions without that GUID and label, but you can't depend on it, and it is allowed to check other partitions that have file systems it understands, but you can't depend on that either.↩

5. Or whatever file name is appropriate on your architecture.↩

6. I'm using backslashes for paths seen by code that's running under UEFI, and forward slashes for code running under Linux, because that's what those parts of the system typically use.↩

Posted by Peter Jones Feb 6th, 2014 EFI System Partition, Linux, UEFI, shim

Tweet

« UEFI Binary Signature Alignment Requirements Secure Boot — Fedora, RHEL, and shim upstream maintenance: government involvement or lack thereof »

Recent Posts

- Secure Boot — Fedora, RHEL, and Shim Upstream Maintenance: Government Involvement or Lack Thereof
- Secure Boot — Fedora, RHEL, and Shim Upstream Maintenance: Government Involvement or Lack Thereof
- Secure Boot — Fedora, RHEL, and Shim Upstream Maintenance: Government Involvement or Lack Thereof
- Secure Boot — Fedora, RHEL, and Shim Upstream Maintenance: Government Involvement or Lack Thereof
- Secure Boot — Fedora, RHEL, and Shim Upstream Maintenance: Government Involvement or Lack Thereof

GitHub Repos

- syslinux

my syslinux development tree

- shim-review
- fedora-kernel

  just a devel branch to share

- gnu-efi

  My gnu-efi development tree.

- objectify

  quick tooling for .obj mesh files

- binutils

  oy

- edk2

  branch of tianocore edk2

- git-toys

just some random git hackery

- puzlp
- xword

  my own copy of an old version of debian's xword package

- grub2-fedora-old

  Branch for keeping track of fedora changes to grub2

- abyss

  This is an abysmal repository. It has nothing.

@vathpela on GitHub