Open Firmware

Recommended Practice:

Generic Names

Version 1.0 (draft)

March 1, 1996 7:15 am

Published by the Open Firmware Working Group

This document is a voluntary-use recommended practice of the Open Firmware Working Group. The Open Firmware Working Group is an ad hoc committee composed of individuals interested in Open Firmware as defined by IEEE 1275-1994, related standards, and their application to various computer systems.

The Open Firmware Working Group is involved both in IEEE sanctioned standards activities, whose final results are published by IEEE, and in informal recommendations such as this, which are published on the Internet at:

```
http://playground.sun.com/pub/1275
```

Membership in the Open Firmware Working Group is open to all interested parties. The working group meets at regular intervals at various locations. For more information send email to:

```
p1275-wg@risc.sps.mot.com
```

Revision History

Version 1.0 First version of document.

1. Introduction

1.1. Purpose

The rules for device naming described in Reference [1] attempt to simultaneously accomplish two objectives:

- To provide a human-readable identifier for use in device paths.
- To provide a unique identification of the device's detailed programming model to allow operating systems to determine which driver to use.

These objectives conflict with one another. For human use within a device path, a brief name that suggests the device's function is preferable. For use in determining an appropriate operating system device driver, an explicit, unique name that includes the manufacturer name and the detailed programming model information is preferable. Attempting to accomplish both objectives with a single name often results in a name that is unsatisfactory for either purpose.

Historically, implementations have often adopted a mixed approach, using brief names for built-in devices and verbose names for plug-in devices. This approach has the problem that the name of a particular device may be different depending on whether the device is built-in or plug-in. Also, because of the tension between the conflicting objectives, different manufacturers chose different trade-offs, some sacrificing uniqueness in favor of human readability and others making the opposite choice.

There is no inherent reason why a single property (`"name"`) must be used for both purposes. The `"compatible"` property already participates in the OS driver selection process, and if the first component of that property's value is an explicit, unique name, precise driver matching will result, even if the `"name"` property's value is a brief *generic* name (e.g. "disk").

Furthermore, the use of *generic* names does not defeat the purpose of unambiguously choosing a a particular device within the device tree through the use of a device path. The fundamental means for ensuring unambiguous node names is the unit-address component (the portion after the "@" character, which matches the first entry of the `"reg"` property's value). The driver-name component (which matches the `"name"` property's value) is inherently unreliable for the purpose of precisely choosing a particular device, due to the possibility of multiple identical devices at a given level of the device tree. Since, in the most general case, the unit-address component must be used to distinguish two devices of the same type, the increased probability of *name space collisions* that would result from the use of *generic* names does not cause loss of functionality. Indeed, it is not unreasonable to think that some users might prefer to distinguish two display devices by the device paths:

/pci/display@2 and /pci/display@4

as compared to:

/pci/IBM,v915 and /pci/number9-723

As already noted, OS driver selection software almost certainly prefers to have the explicit information contained in, for example, *IBM,v915*, but the software can get that information from the `"compatible"` property as easily as from the "name" property.

1.2. Scope

This recommended practice changes the naming conventions for device nodes, affecting plug-in cards, Open Firmware system implementations, and client programs.

2. References and Definitions

2.1. References

[1] *IEEE Std 1275-1994, IEEE Standard for Boot(Initialization Configuration)Firmware:Core Requirements and Practices*, published by IEEE.

[2] *PCI Local Bus Specification*, Revision 2.1, June 1, 1995

2.2. Definitions

3. *Generic* Names

Because of the difficulties that result from using *name* for two conflicting purposes, and since that dual use is unnecessary, the working group recommends the following guidelines for future devices:

Guideline 1: `"name"` property values should be *generic*, reflecting the device's function, but not necessarily its precise programming model. If appropriate, the value should be one of the following listed below:

- disk
- tape
- pci
- vme
- sbus
- scsi
- token-ring
- isa
- keyboard
- display
- mouse
- audio
- ethernet
- timer
- memory
- printer
- serial
- rtc
- nvram
- scanner
- (others?)

The working group may define additional *generic* names in future recommended practice documents or binding documents. For devices that do not fit in any existing generic category, the developer may request that the working group establish a new generic name, or may use an explicit name prefaced with a manufacturer name.

This recommendation supersedes the stipulation in Reference [1] that `"name"` property values for plug-in devices must be prefaced with a manufacturer name. The intended purpose of that requirement (which is to allow precise OS driver selection) is fulfilled by Guideline 2, below.

Guideline 2: The `"compatible"` property should be present. Its first component should be an explicit, unique name that identifies the device precisely enough to be able to infer the detailed programming model from that name. The format of that explicit name is *manufacturer,model*, as specified by Reference [1] for the `"name"` properties of plug-in devices (see Reference [1], page 176, lines 16-27).

4. Affect on OS Driver Selection Code

An operating system that implements the driver searching technique that Reference [1] recommends for the `"compatible"` property (see Reference [1], page 141, lines 10-14) is likely to require no changes as a result of this recommended practice. If a `"name"` property has a generic value, the search for a driver matching that generic name is likely to fail, but the next step of the search (using the `"compatible"` property) will succeed, finding the same driver that would have been found had the explicit name been in the `"name"` property.

It is possible that the operating system will find a driver matching the generic name, and that said driver will not be the correct one. However, this is not a new problem, because generic names have already been used in the past for built-in devices. Consequently, an operating system that does not already have a mechanism for resolving or avoiding such *false matches* is likely to have problems eventually, with or without the proliferation of generic names.

The following suggests some possible techniques for dealing with such *false generic matches*.

a) In collections of OS drivers, avoid the use of generic names for the drivers themselves. For example, it is generally unwise to name a driver *ethernet*, since there are many different ethernet adapters with different programming models. Using the generic name *ethernet* to identify only one such driver is presumptuous.

b) Separate the OS's name spaces for drivers for *real* hardware devices and *pseudo-drivers* (collections of support routines that are used by *real* drivers). Some operating systems load such pseudo-drivers using a mechanism similar to the mechanism used for *real* drivers. Pseudo drivers often perform g*eneric* functions that apply equally well to, for example, all ethernet adapters, independent of their low-level programming models. Consequently, it is reasonable to use generic names (e.g. *ethernet*) for such pseudo-drivers. Separating the name spaces of *real* drivers and pseudo-drivers avoids false matches from generic device names to generic psuedo-driver names.

c) Make the OS's driver search mechanism depend upon the device's parent. In other words, separate the OS's driver name spaces so that drivers for devices that attach to, for example, PCI bus can be distinguished from those that attach to, for example, ISA bus. This reduces the range over which *false matches* can occur.

d) For cases where false matches are unavoidable (for example, if there is an existing driver with a generic name that must be retained for backwards compatibility) allow the drivers that can be incorrectly matched the possibility of rejecting the match. One technique for doing so is to for the driver to inspect the compatible property to ensure that it is appropriate. Another common technique is to have the generic driver *probe* the hardware to see it it behaves as expected (although this technique can cause problems).

5. Existing Devices

Existing devices with explicit names need not change. The recommended search techniques continue to work correctly with such devices.

6. PCI Name Properties

Instead of creating their childrens device's name properties from the vendor and device ID's, PCI bus device nodes should create them from the class code, according to the following table(See Reference [2], Appendix D, Class Codes):

Table 1. PCI Naming Convention Using Class Codes

Base Class	Sub-Class	Interface	Meaning	Generic Name
00h	00h	00h	All currently implemented devices except VGA-compatible device.	
00h	01h	00h	VGA-compatible device.	
01h	00h	00h	SCSI bus controller	
01h	01h	xxh	IDE controller	
01h	02h	00h	Floppy disk controller	
01h	03h	00h	IPI bus controller	
01h	04h	00h	RAID controller	
01h	80h	00h	Other mass storage controller	
02h	00h	00h	Ethernet controller	
02h	01h	00h	Token Ring controller	
02h	02h	00h	FDDI controller	
02h	03h	00h	ATM controller	
02h	80h	00h	Other network controller	
03h	00h	00000000b	VGA-compatible controller	
03h	00h	00000001b	8514-compatible controller	
03h	01h	00h	XGA Controller	
03h	80h	00h	Other display controller	
04h	00h	00h	Video device	
04h	01h	00h	Audio device	

Table 1. PCI Naming Convention Using Class Codes

Base Class	Sub-Class	Interface	Meaning	Generic Name
04h	80h	00h	Other multimedia device	
05h	00h	00h	RAM	
05h	01h	00h	Flash	
05h	80h	00h	Other memory controller	
06h	00h	00h	Host bridge	
06h	01h	00h	ISA bridge	
06h	02h	00h	EISA bridge	
06h	03h	00h	MCA bridge	
06h	04h	00h	PCI-to-PCI bridge	
06h	05h	00h	PCMCIA bridge	
06h	06h	00h	NuBus bridge	
06h	07h	00h	CardBus bridge	
06h	80h	00h	Other bridge device	
07h	00h	00h	Generic XT-compatible serial controller	
07h	00h	01h	16450-compatible serial controller	
07h	00h	02h	16550-compatible serial controller	
07h	01h	00h	Parallel port	
07h	01h	01h	Bidirectional parallel port	
07h	01h	02h	ECP 1.X compliant parallel port	
07h	80h	00h	Other communications device	
08h	00h	00h	Generic 8259 PIC	
08h	00h	01h	ISA PIC	
08h	00h	02h	EISA PIC	
08h	01h	00h	Generic 8237 DMA controller	
08h	01h	01h	ISA DMA controller	
08h	01h	02h	EISA DMA controller	
08h	02h	00h	Generic 8254 system timer	
08h	02h	01h	ISA system timer	
08h	02h	02h	EISA system timers(2 timers)	
08h	03h	00h	Generic RTC controller	
08h	03h	01h	ISA RTC controller	
08h	80h	00h	Other system peripheral	
09h	00h	00h	Keyboard controller	

Table 1. PCI Naming Convention Using Class Codes

Base Class	Sub-Class	Interface	Meaning	Generic Name
09h	01h	00h	Digitizer (pen)	
09h	02h	00h	Mouse controller	
09h	80h	00h	Other input controller	
0Ah	00h	00h	Generic docking station	
0Ah	80h	00h	Other type of docking station	
0Bh	00h	00h	386	
0Bh	01h	00h	486	
0Bh	02h	00h	Pentium	
0Bh	10h	00h	Alpha	
0Bh	20h	00h	PowerPC	
0Bh	40h	00h	Co-processor	
0Ch	00h	00h	Firewire(IEEE 1394)	
0Ch	01h	00h	Access bus	
0Ch	02h	00h	SSA	
0Ch	03h	00h	Universal Serial Bus(USB)	
0Ch	04h	00h	Fibre Channel	

The `"vendor-id"` and `"device-id"` properties should continue to reflect the same explicit information as before.

For PCI devices without FCode drivers, the PCI bus node should automatically create a `"compatible"` property with a single entry whose value reflects the vendor and device ID's, as previously specified for the name property.

For devices with FCode drivers, that FCode device driver *shall* create a `"compatible"` property whose first entry uniquely identifies the functions programming model, as previously defined for FCode-generated `"name"` properties. That `"compatible"` property may have additional entries as appropriate. If the device's programming model is not entirely defined by the characteristics of the chip that is directly connected to the PCI bus (for example, if there are additional programmable chips *behind* that directly-connected chip), then that first entry should indentify the complete programming model for the device (i.e. the vendor and model number for the entire card), not just the programming model of the directly-connected chip. In any case, the `"compatible"` property value *shall* also contain a string of the form *"pciVVVV,DDDD"*, identifying the directly-connected chip. *VVVV* and *DDDD* are lower-case ASCII hexadecimal numbers, without leading zeroes, representing the Vendor ID and Device ID of the chip that is directly connected to the PCI bus. The `"compatible"` property may have additional entries as appropriate.

Examples:

Preferred: *IBM,v915 pci5333,88b0 vga*
Acceptable: *pci5333,88b0 vga*

Note: As defined by IEEE 1275, the value of the "compatible" property is a list of strings encoded as with "encode-string", so the property value in the *Preferred* example above could be generated with:

"IBM,v915" **encode-string**
"pci5333,88b0" **encode-string encode+**
"vga" **encode-string encode+**
"`compatible`**"** property

The resulting prop-encoded array consists of a series of null-terminated strings concatenated one after another.

Note: It is better to identify the function as a whole, instead of identifying only the *chip* that is directly connected to the PCI bus (e.g.: use *IBM,v915,* instead of *S3,928)*, because, there may be numerous other display adapters that use the same graphics chip, but which are not entirely compatible due to the presence of different support chips like DAC's and programmable clock generators.