OpenBoot Dropin Modules

Michael Milendorf
Sun Microsystems, Inc.
One Network Drive, Burlington, MA 01803 USA
michael.milendorf@sun.com

Abstract

Firmware is the ROM-based software that controls a computer between the time it is turned on and the time the primary operating system takes control of the machine. OpenBoot is a Sun Microsystems implementation of IEEE Standard 1275-1994 for Boot Firmware: Core Requirements and Practices. A dropin module is a file or a data structure that is stored on various devices, such as flashprom, eeprom, nvram, floppy diskette, disk, cdrom and network. The use of dropin modules in OpenBoot firmware provides an elegant vehicle for packaging and delivering firmware device drivers and diagnostics into RAM. Firmware can optionally load and execute dropins during machine initialization, configuration and testing. The paper describes Dropin Technology and its various applications.

1 Assumptions

The paper makes an assumption that the reader is somewhat familiar with the concept of Open Firmware. The best known document on the topic is **IEEE 1275-1994 Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices**, published by the Institute of Electrical and Electronics Engineers, Inc., 345 East 47th street, New York, NY 10017-2394, USA. Dropins are not defined by the standard and it is Sun Microsystems technology.

2 ROM-resident Dropin Solution

A dropin concept was introduced in early OpenBoot by Mitch Bradley (1991). A dropin was defined as a package which was placed in unused space at the end of a machine's boot PROM after OpenBoot firmware code. A dropin could contain data or executable code in any number of different formats.

One application for dropins was the support of slightly different machines using the same OpenBoot binary image. For example, a SPARC-compatible vendor might design a workstation which was very similar to a Sun SPARCstation, but which had additional on-board I/O device.

This machine could use the standard Sun SPARCstation boot PROM, with an additional dropin added to support the new device. Dropins provided many benefits in this situation. The vendor only needed to license the binary PROM from Sun, not the complete OpenBoot source code. In addition, including a dropin was easier and less error-prone than making source modifications, and the use of dropins enhanced reliability and reduced development time.

Another useful way of using dropins was adding OpenBoot driver support for plug-in, non-self-identifying devices. If a plug-in device didn't have a PROM, or didn't provide an FCode driver on it, OpenBoot could provide a dropin driver for such a device.

Executable dropins could contain code in one of the following four formats:

- FCode
- Forth
- Standalone Program
- Machine Code

Dropins didn't contain executable code only, they may contained data as well. This was useful for storing data which was necessary for firmware operation, but which was optionally or infrequently accessed, for example data for different fonts or for keyboard support.

Dropins were identified by their names with some names known to OpenBoot. Dropins with known names were automatically called at various points in the startup process. Multiple dropins with the same name were executed in the order that they appeared in the PROM.

Each dropin began with a header of the following fixed length fields: *Magic Number* field, *Image Size* field, *Checksum* field, *Reserved* field and *Dropin Name* field. The last *Dropin Image* field encapsulated the module data itself.

The following two OpenBoot interfaces were written to access dropins:

find-drop-in (name-addr,len -- false | dropin-addr size true)

> **find-drop-in** attempts to locate the first dropin in the boot ROM whose name matches the argument string. If successful, it returns the dropin ROM *address*, its *size* and a *true* flag on the stack. If there is no dropin by that name in the boot ROM, a *false* flag is returned on the stack. The pointer returned by **find-drop-in** refers to the boot ROM itself.

do-drop-in (name-addr,len --)

> **do-drop-in** executes all dropins whose names match the argument string, in the order that they appear in the ROM. If no dropins are found nothing happens. Dropins may be called from other dropins.

3 General Device-independent Dropin Solution

The original goal for dropins was to make it possible for vendors of SPARC clones to make small changes to the startup code without having to change OpenBoot source code. In keeping with that simple goal, the dropin design was relatively simple. Over the years dropins started to be used more extensively and the design became inadequate for some of the newer uses. Today, one of the new dropin applications is to package firmware device drivers and firmware diagnostics as boot ROM dropins. The whole I/O subsystem (all the drivers) is now written in

FCode (versus Forth in early days) and is packaged as a set of individual dropins. All firmware diagnostics are also written in FCode and packaged as a set of individual dropins. Such an arrangement affords the OpenBoot ROM a much more dynamic structure and function similar to the Solaris Operating System loadable module concept.

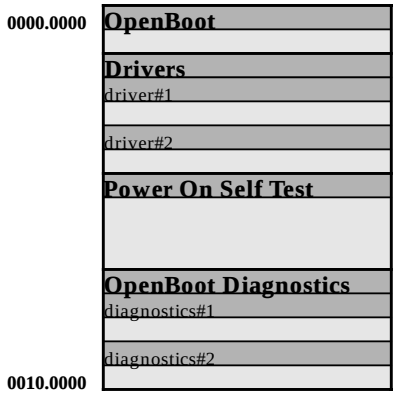
The proliferation of dropin modules provoked the need to use boot ROM dropin data compression and decompression. Now the pointer returned by **find-drop-in** referred to the RAM address.

Up to now, dropins were stored only in the boot ROM. At some point that was assessed as an unnecessary restriction. It was discovered that dropin files could be kept not only in the main boot ROM (*flashprom*) but on any storage device, such as motherboard *eeprom*, *nvram* and even media devices, like *floppy diskette*, *disk*, *cdrom* and *network*.

The underlying mechanism to support the implementation of generic device-independent dropin solution is supported by OpenBoot implementation of **IEEE 1275-1994** standard interfaces for these devices. It turns out that there is no need to invent any new non-standard interfaces (like old **find-drop-in** and **do-drop-in**). The standard device methods **open**, **close**, **size** and **read** represent a complete working interface for the dropin modules. For most firmware device drivers (*floppy*, *disk* and *network)* these methods are already defined.

An exception was the boot *flashprom* device, which of course remains the primary source of dropin images, and for which a new support driver was written, containing **open**, **close**, **size** and **read** methods. This broader approach to the dropin devices called for further improvement and suggested to implement a multi-level embedded dropin structure for the *flashprom* device to make it adherent to file systems on other devices. Now the path for ***driver#1*** in the *flashprom* device could be represented as ***/flashprom@10,0:, | Drivers | driver#1.***

Figure 1 OpenBoot FLASHPROM



Each dropin contains the **header** and the **data** itself, the **data** could be another dropin.

OpenBoot flashprom partitioning:

Several level-1 dropins: **OpenBoot** (contains assembly and Forth boot up code), **Drivers** (contains level-2 FCode device drivers), **Power On Self Tes**t (contains assembly test code), **OpenBoot Diagnostics** (contains level-2 FCode selftest diagnostics).

Further one could use a standard file system for the *flashprom* device. If this were done it would allow Sun Microsystems OpenBoot firmware to switch from a private dropin format to one of the standard file formats for *flashprom* devices.

Whatever the final implementation for the *flashprom* device will be, here is the device-independent generic solution to support dropin modules (files) in OpenBoot firmware:

Have the standard set of **open**, **close**, **size** and **read** methods for every device which wants to support dropin modules. The methods are described in **IEEE 1275-1994** and do the following:

open (-- flag)

Prepare this device for subsequent use. Typical behavior is to allocate any special resource requirements it needs, map the device into virtual address space, initialize the device and perform a brief “sanity test” to ensure that the device appears to be working correctly. Return *true* if this **open** method was successful, *false* if not.

close (--)

Close this previously opened device. Restore the device (which has been previously **open**ed to its “not-in-use” state. Typical behavior is to turn off the device, unmap it, and deallocate any resources that were allocated by **open**.

size (-- d)

Return the size of the device in bytes. If the size can’t be determined return -1.

read (addr len -- actual)

Read device into memory buffer; return *actual* byte count. Read at most *len* bytes from the device into the memory buffer beginning at *addr*. Return *actual*, the number of bytes actually read. If *actual* is zero or negative, the read operation did not succeed.

In addition to those interfaces create the dropin support package consisting of four methods: **open**, **close**, **size** and **read**. Create the software **/dropins** package in OpenBoot device tree for those methods to provide a calling interface to the methods of the specific hardware devices (*flashprom*, *floppy*, etc.).

Code Example 1

```
: open  ( -- ihandle ) my-args ascii : left-parse-string $open-package ;
: read  ( addr len ihandle -- actual )“ read” rot $call-method ;
: size  ( ihandle -- d )              “ size” rot $call-method ;
: close  ( ihandle -- )  close-package ;
```

The device path and the dropin name are provided as argument string (through **my-args**) to the **open** package method.

Summarizing, the complete dropin support environment in OpenBoot firmware consists of:

- **/dropins** device tree package containing **open**, **close**, **size** and **read** methods.
- Standard **IEEE 1275-1994** firmware drivers for the devices (implementing **open**, **close**, **size** and **read** methods) which are intended to be used as a storage source for the dropin modules.

In case of *floppy*, *disk*, *cdrom* or *network* devices, dropin modules use the standard Unix file system format. In case of ROM-class devices (like *prom*, *eeprom*, *flashprom* or *nvram*) we can use OpenBoot format for dropins as described earlier in the paper, or implement the standard file system on those devices.

4 Firmware Use of the Dropins

With the availability of **/dropins** package, the functionality of accessing dropins can be rewritten based on the standard interfaces in the following way. **find-drop-in** now takes the full device pathname, including information about dropin device and dropin name and simply **open**s and **read**s the dropin device.

Dropins can be placed on any storage device which provides the standard interfaces: **open**, **close**, **size** and **read**. A successful **open** method sets the internal pointer to the dropin data, and enables **size** method to return the length of the dropin image in bytes. Then, provided with the address of the memory buffer in RAM, the **read** method moves returned number of bytes from the dropin source storage device into the memory buffer at the given address. The **close** method restores the device to "not-in-use" state and releases resources.

Code Example 2 Definitions for **find-drop-in**, **do-drop-in**

```
: find-drop-in ( name-addr,len -- false | dropin-addr size true )
   " /packages/dropins" $open-package   ( device-ihandle pkg-ihandle)
   >r >r r@  if                         ( )
      " size" r@ $call-method dup       ( dropin-size dropin-size)
      alloc-mem dup rot                 ( RAM-addr RAM-addr dropin-size)
      " read" r> $call-method           ( RAM-addr actual-size )
      " close" r> close-package true    ( dropin-addr size true )
    else                                ( )
      r> r> 2drop false                 ( false )
   then
;

: free-drop-in ( dropin-addr size -- ) free-mem ;

: do-drop-in ( name-addr,len -- )
   find-drop-in  if
      over 1 byte-load
      free-drop-in
   then
;
```

Below are two examples of **find-drop-in** and **do-drop-in** use. By default, OpenBoot uses *flashprom* as a dropin storage device and calls **do-drop-in** and **find-drop-in** as part of it start-up sequence:

Code Example 3 Use of **find-drop-in**, **do-drop-in**

```
ok " /flashprom:,|Drivers|pci1000,f" do-drop-in
ok " /disk@0,0:a,|Drivers|fonts"     find-drop-in
```

Depending on the storage device, dropin modules may or may not be compressed. *Flashprom* dropins are always compressed and thus it is a job of *flashprom* device **read** method to decompress the dropin, before moving the image to a specified memory address.

5 Diagnostics Use of the Dropins

Another interesting application of dropins lies in the firmware diagnostics realm. The primary **IEEE 1275-1994** interface for diagnostics in firmware is the **test** command:

test (“device-specifier<eol>” --)

> Invoke the *selftest* routine for the specified device. If the device node specified by *device-specifier* has a *selftest* method, invoke it with *execute-device-method*. Otherwise display an error message.

According to this original specification of the **test** method, the *selftest* routine must already be present in memory for the **test** command to invoke it. OpenBoot implementation of **IEEE-1275-1994** extends the semantics of the **test** command to do the following:

test (“device-specifier<eol>” --)

> Invoke the *selftest* routine for the specified device. If the device node specified by *device-specifier* has no *selftest* method, attempt automatic loading of the *selftest* method. If the device node has *selftest* method, or if it was added as a result of automatic loading, invoke it with *execute-device-method*. If *selftest* is absent and if automatic loading did not succeed display an error message.

The automatic loading of *selftest* method can be elegantly implemented if *selftest* routine is packaged as a dropin module.

Each hardware device in OpenBoot is represented by a device tree node. Each node has a set of properties. The *name* property is a text string consisting of a sequence of up to 31 characters. The *compatible* property is a list of text strings, each string consisting of up to 31 characters. The *name* property represents the generic name of the device, for example ***scsi***. The *compatible* property represents the list of specific names, from the most-specific to the least-specific, for example {***pci1000,f pciclass,001000***}. If an entry of the *compatible* or *name* property of the device node is selected as a dropin name, and **dropin-device** configuration variable contains the list of possible dropin devices (pathnames), the following algorithm (as implemented in OpenBoot) performs automatic loading of the dropin *selftest* package:

Figure 2 Automatic Loading Algorithm

A: *For every* ***name*** *entry (from the most-specific to the least-specific), taken from {****compatible****,* ***name****} list of device tree node properties, indicated by* ***device-specifier*** *(arguments to* **test** *command) Do:*

> ***B:*** *For every device* ***path*** *(parsing left to right) in the* ***dropin-device*** *configuration variable Do:*
>
> 1. *Append* ***name****, which is a dropin name, to the* ***path*** *as extracted from the* ***dropin-device****, effectively creating* ***pathname*** *argument string.*
> 2. *Call* ***“ pathname” “ dropins” $open-package*** *(-- flag).*
> 3. *If* ***false*** *flag is returned (dropin not found) goto* ***B****.*

4. *Otherwise the dropin was found. Call the **size** method of **/dropins** package and allocate **size** bytes of RAM memory. Call the* **read** *method of **/dropins** package, to read the dropin into RAM. Prepare device node, indicated by **device-specifier** to the evaluation of FCode and execute* **1 byte-load** *which effectively interprets FCode from the dropin image in RAM (the dropin package must contain all support methods and definition for the **selftest** method). Release resources, and re-establish prior firmware environment.*

5. *Now, **selftest** routine is present in the device node, indicated by **device-specifier**. Invoke **selftest** method with **execute-device-method**. Exit to **D**.*

Loop;

Loop;

***C:** Display an error message:* ***There is no selftest method for the device**.*

***D:** End.*

Here is an example of how the algorithm works at the user level:

Lets say we are about to test the scsi device, which is a *Symbios* disk. The OpenBoot node for the *scsi* device is */pci@1f,4000/scsi@2* with name property set to "***scsi***" and *compatible* property set to "***pci1000,f_glm_pciclass,001000***".

OpenBoot has a diagnostic dropin *selftest* package (in the *flashprom*) to test a wide range of *scsi* devices. Also there is a newer diagnostic (on a *floppy* diskette), designed to diagnose the specifics of the *Symbios pci1000,f scsi* device in addition to its more generic set of scsi functions.

The less-specific diagnostic is stored in *flashprom* as **|*OpenBootDiagnostics*|scsi** dropin.

The more-specific diagnostic is stored on a *floppy* diskette as **|*Diagnostics*|*pci1000,f*** file.

Code Example 4

```
ok setenv dropin-device flashprom
ok test /pci@1f,4000/scsi@2
Starting Generic SCSI Selftest
Test is completed
```

In the *Example 4*, **test** searches only *flashprom* device and loads **scsi** dropin. Now we reset the machine and execute the following commands:

Code Example 5

```
ok setenv dropin-device flashprom floppy
ok test /pci@1f,4000/scsi@2
Starting Symbios SCSI Selftest
Test is completed
```

In the *Example 5*, **test** searches both the *flashprom* and the *floppy* diskette. Because **test** searches from the most-specific to the least-specific name, it picks the ***pci1000,f*** dropin stored on the *floppy* diskette before the **scsi** dropin stored on the *flashprom* is considered. The result is that the more-specific diagnostic code ends up being loaded and executed. In all cases **test** will automatically find, load and run the best suitable diagnostic for the device.

Another example could use a plug-in card with no *selftest* method or FCode whatsoever. In such a case, OpenBoot can include the *selftest* dropin for a plug-in device in the motherboard *flashprom*. Or the *selftest* dropin file could be stored anywhere on the *network*, or on a *floppy* diskette, or on any suitable storage device. Now, even if the plug-in device doesn't contain FCode, OpenBoot at probe time of the machine will build a device node entry for the card and will construct a *compatible* property (for PCI devices). **test** can now diagnose a device without selftest or FCode for it.

6 Network as a Dropin Device

There are few issues associated with selecting *network* as a dropin-device:

First of all *network* is a slow response device and there is no way to distinguish between "not ready yet" condition and "file not found" condition. Therefore, the only way to support dropins on a *network* is to support a time-out parameter (number of ***retries***) argument in a *network* device path (an **IEEE 1275-1994** recommended practice currently not supported in OpenBoot). If number of ***retries***, specified in the device path argument expired and an acknowledgment is not received, the *network* **open** method returns a *false* flag (dropin file not found).

Code Example 6

```
ok " /network@0,0:,pathname,,,,retries" do-drop-in
```

Another problem is that the size of the file is not provided in ***T****rivial* ***F****ile* ***T****ransfer* ***P****rotocol* (***TFTP*** - OpenBoot *network* protocol) before the actual *network* **read** is completed. So, the **size** definition of /**dropins** package must be modified to return a fixed *n* bytes if **size** method is not found for the device (*network*). Or ignore **size** altogether and simply provide a fixed *n* bytes as a size instead. If that approach is taken, it would be wasteful to allocate too much memory region. Instead use system allocated memory region (**load-base**) which is known to be larger than *n* bytes in size.

7 Summary

This paper discussed the history and development of OpenBoot Dropin Technology. Dropins are data modules or files, or executable modules or files which could optionally be loaded into memory from different kinds of computer storage devices. Dropins can be loaded and executed by **open**ing, and **read**ing **/dropins** software package. Dropin device *path* and dropin *name* are provided as a single *pathname* text string argument to the **open** method. Dropins Technology represents a compact, modular, device-independent way of packaging firmware components and delivering them into system memory. The implementation of Dropins Technology is elegant and simple, and showing the advantages Sun Microsystems, Inc. gains, using standard implementation of **EEE 1275-1994** for its boot firmware.