Device Support Extensions to:

IEEE Std 1275-1994

Standard for Boot

(Initialization, Configuration)

Firmware

Revision: 0.5 DRAFT

Date: February 28, 1996

Purpose of this Device Support Extensions binding

This document specifies the application of Open Firmware to Device Support Extensions, including requirements and practices to support unique firmware specific to the support of device extensions. The core requirements and practices specified by Open Firmware must be augmented by device-specific requirements to form a complete specification for the firmware implementation for Device Support Extensions. This document establishes such additional requirements pertaining to the device extensions and the support required by Open Firmware.

Task Group Members

The Device Support Extensions binding team members were the following:

Mitch Bradley, FirmWorks

Jordan Brown, SunSoft

Bob Coffin, editor, IBM

Ron Hochsprung, Apple

David Kahn, Sun Microsystems

Mike Segapeli, IBM

Trademarks

The following terms, denoted by a registration symbol (®) or trademark symbol(™) on the first occurrence in this publication, are registered trademarks ot trademarks of the companies as shown in the list below:

Trademark	Company

Revision History

Revision 0.1	04-05-95	Initial draft, split from the original PReP binding.
Revision 0.2	07-05-95	Made some editorial changes. Added aliases node properties.
Revision 0.3	08/17/95	Added Sections ‘Control Sequences for Common Special Keys’, ‘Additional Requirements for SCSI Devices’, and ‘Additional Requirements for Block and Byte Devices.
Revision 0.4	11/09/95	Changed NVRAM ‘open’ and ‘close’ methods verbage. Added requirements for SCSI Devices; properties and methods.
Revision 0.5	02/26/96	Changed format for *Device Support Extension binding*; added ‘Trademarks’, ‘Revision History’ Sections plus ‘Table-of-Contents’. Modified Sound Device for common data format support. Added NVRAM ‘size’ property and method.

Table of Contents

1. Overview 7
2. References 7
3. Keyboard devices 7
    3.1 "keyboard" device standard properties 8
    3.2 "keyboard" device methods 8
    3.3 Abort key sequence for keyboard devices 8
    3.4 Control Sequences for Common Special Keys 8
4. Pointing devices 9
    4.1 "mouse" device standard properties 9
    4.2 "mouse" device methods 9
5. Real Time Clock (RTC) Device 10
    5.1 "rtc" device standard properties 10
    5.2 "rtc" device methods 10
6. Sound Device 10
    6.1 "sound" device standard properties 10
    6.2 "sound" device standard methods 10
7. NVRAM Device 11
    7.1 NVRAM properties 11
    7.2 NVRAM methods 11
8. Parallel port device 12
    8.1 Parallel port properties 12
    8.2 Parallel port methods 12
9. Additional Requirements for SCSI Devices 12
10. Additional Requirements for Block and Byte devices 12
11. Additional Requirements for Block Devices 12
12. /aliases Node Properties 13

1. Overview

This document specifies the application of *IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware, Core Practices and Requirements*[1] to a variety of common peripheral types. It is intended that Open Firmware implementations supporting these device types will supply the methods and properties described herein.

2. References

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

[2] *ISO-639*, *Code for the representation of names of languages*, published by International Organization for Standardization.

[3] *ANSI/IEEE X3.215-1994, Programming Languages -- Forth.*

3. Keyboard devices

Open Firmware does not have a specific device class for keyboards; instead, keyboards are instances of the `"serial"` device class. However, certain features of keyboards (i.e., the ability to re-map keys, etc.) make it desirable to implement them as a separate device class.

In general, keyboard devices produce a hardware *scan-code* that is specific to the type of keyboard. These scan-codes are then mapped via software to produce the character code for the key, taking into account the state of "modifier" keys (e.g., Shift, Control) and the keyboard layout. The mapping of scan-codes to character value depends upon the keyboard layout; this is dependent upon the language that is being supported by the keyboard. E.g., the layout of keys for a French keyboard is different than that for an English keyboard.

For purposes of localization, it is necessary for the scan-code conversion to be controlled by the software, including Open Firmware (via the Client Interface). This section adds mechanisms to allow client programs to alter the scan-code conversion, based upon the keyboard (language) layout.

The language-specific layout is specified by means of a 2-character abbreviation, as described by [2]. The following languages may be supported:

CS	Czech
DA	Danish
NL	Dutch
EN	English (default)
FI	Finnish
FR	French
DE	German
HU	Hungarian
IT	Italian
NO	Norwegian
PL	Polish
PT	Portuguese
SK	Slovak
ES	Spanish
SV	Swedish

For testing of keyboard devices, additional low-level methods are defined for `"keyboard"` devices. **`get-scancode`** will read the next available scan-code from the keyboard; **`scancode->char`** will perform character mapping, using the same conversion as would normal **`read`**s. This enables test software to test for a specific character to terminate the test without having to know the scan-code to character translations.

3.1. "keyboard" device standard properties

"device_type" **S**

Standard *property*; the value of this property for keyboard devices *shall* be "keyboard".

"language" **S**

Standard *property*, encoded as with **encode-string**, that indicates the current scan-code to character conversion, based upon the language's keyboard layout. This value indicates the language (i.e., keyboard layout scan-code conversion) to which the keyboard driver is currently set.

3.2. "keyboard" device methods

In addition to the Open Firmware standard **open**, **close** and **read** methods, the following methods *shall* be supported by an Open Firmware implementation of a "keyboard" device. Note that the **open** routine can take an optional argument which specifies the language (i.e., **scancode->char** mapping) to be used.

get-scancode (msecs -- scancode true | false) **M**

This method returns the "raw" scan-code value of the next key alteration. msecs is the number of milliseconds to wait for a keystroke before reporting failure; a value of 0 implies wait until keystroke. If this timeout expires before a keystroke is read, **false** is returned. Otherwise, the scancode and **true** are returned.

scancode->char (scancode -- char true | false) **M**

Using the scancode, perform translation to a character. If the scancode represents a modifier key (e.g., Shift), no translation will be available; in this case, **false** is returned. If the scancode represents a translated character, char and **true** are returned.

set-language (str len -- true | false) **M**

If the keyboard driver can support the requested character-set, it ***shall*** set its "language" property to the value specified by str,len, and return a value of **true**. If it can not support the requested language, no change of "language" is made and a value of **false** is returned.

The reason for adding a special call, instead of just using the **property** (setprop) call, is to allow the device to "filter" the request. I.e., a scan-code conversion may not be available for the requested language. This call allows the device driver to change the property only if it can support the requested mapping.

3.3. Abort key sequence for keyboard devices

For PC-101 keyboards, the abort sequence (see *install-abort* in the IEEE 1275 glossary) *shall* be Ctrl-Brk.

3.4. Control Sequences for Common Special Keys

The following table specifies the values that should be returned in the keyboard buffer when certain keys are pressed.

Table 1. Control Sequences for Common Special Keys

Sequence	Key legend
CSI[a] A	up arrow
CSI B	down arrow
CSI C	right arrow
CSI D	left arrow
CSI H	home

Sequence	Key legend
CSI @	insert
CSI P	delete[b]
CSI K	end
CSI ?	page-up
CSI /	page-down

Table 1. Control Sequences for Common Special Keys (Continued)

Sequence	Key legend
CSI O[c] P	function 1
CSI O Q	function 2
CSI O w	function 3
CSI O x	function 4
CSI O t	function 5
CSI O u	function 6

Sequence	Key legend
CSI O q	function 7
CSI O r	function 8
CSI O p	function 9
CSI O M	function 10
CSI O A	function 11
CSI O B	function 12

a. Note: The CSI (Control Sequence Introducer) is equal to the value 09B hex.

b. Note: Certain external devices often return 07F hex when the "delete" key is pressed.

c. The letter 'O'

4. Pointing devices

This class of device covers a broad category of "pointing" devices, the most common embodiment of which is the mouse. These devices typically can generate X,Y coordinates and button-press information on some periodic basis. The following properties and methods are defined for such devices.

4.1. "mouse" device standard properties

"device_type" **S**

Standard *property*; the value of this property for pointing devices *shall* be "mouse".

"#buttons" **S**

Standard *property*, encoded as with **encode-int**, that indicates the number of physical buttons supported by the device. This property can be used to interpret the *buttons* value returned by the **get-event** method.

"absolute-position" **S**

Standard *property*, whose presence indicates that this device supplies absolute X,Y coordinates (e.g., a graphics tablet). Absence of this property indicates that the device supplies relative X,Y position (e.g., a mouse).

4.2. "mouse" device methods

In addition to the Open Firmware standard **open** and **close** methods, the following methods *shall* be supported by an Open Firmware implementation of a "mouse" device.

Pointing devices typically supply data only when an "event" occurs (e.g., the mouse moves or a button is pressed). The following method attempts to obtain an event from the device, reporting whether an event occurred.

get-event (msecs -- pos.x pos.y buttons true | false) M

Standard method for obtaining the next "event" of pointing devices. msecs is the number of milliseconds to wait for an event before reporting failure; a value of 0 implies wait until event. pos.x, pos.y return the positioning information; they are interpreted as unsigned or signed, depending upon the presence or absence of the "absolute-position" property. buttons returns a bit-mask (in the low-order bits) representing any buttons that are pressed; the number of significant bits to examine is defined by the "#buttons" property. The least significant bit corresponds to the primary (usually the left-most) button of the device. The top stack result indicates whether an event was detected within the time-out period.

5. Real Time Clock (RTC) Device

Open Firmware for Real Time Clocks defines the following properties and methods. The representation of time is defined by the TIME&DATE method of the ANS Forth standard [3].

5.1. "rtc" device standard properties

"device_type" **S**

Standard *property*; the value of this property *shall* be "rtc".

5.2. "rtc" device methods

In addition to the Open Firmware standard **open** and **close** methods, the following methods *shall* be supported by an Open Firmware implementation of a "rtc" device.

get-time (-- n1 n2 n3 n4 n5 n6) **M**

Return the current time as the integers n1…n6, where n1 is the second {0…59}, n2 is the minute {0…59}, n3 is the hour {0…23}, n4 is the day {1…31}, n5 is the month {1…12}, and n6 is the year (e.g., 1994).

set-time (n1 n2 n3 n4 n5 n6 --) **M**

Set the current time from the integers n1…n6, where n1 is the second {0…59}, n2 is the minute {0…59}, n3 is the hour {0…23}, n4 is the day {1…31}, n5 is the month {1…12}, and n6 is the year (e.g., 1994).

6. Sound Device

In order to use a sound device within the context of Open Firmware (e.g., "boot beeps"), the following properties and methods *shall* be implemented. To provide a common data format for universal support, Open Firmware *shall* ,by default, pass audio to **write** or return audio from **read** that consists of 8-bit monophonic samples encoded as unsigned linear values, centered at 128.

Note 1: Some implementation of "sound" nodes may need to convert this data into a form acceptable to the audio hardware being supported. It is not anticipted that this is an unreasonable requirement.

Note 2: An implementation may choose to supply additional properties, methods and "open" arguments so as to support more advanced audio capabilties, so long as a default "open" results in _write_ interpreting the standard format.

6.1. "sound" device standard properties

"device_type" **S**

Standard *property*; the value of this property *shall* be "sound".

"#channels"

Standard *property*, encoded as with **encode-int**, that represents the number of channels supported by the device.

"sample-resolution" **S**

Standard *property*, encoded as with **encode-int**, that represents the number of bits of resolution for each sound sample.

"sample-width" **S**

Standard *property*, encoded as with **encode-int**, that represents the number of bytes required for storing a sample.

"sample-rates" **S**

Standard *property*, consisting of an array of integers, each encoded as with **encode-int**, that represents the rates (in hertz) at which this device can be sampled.

6.2. "sound" device standard methods

The following methods *shall* be implemented by a "sound" device.

open **(-- true | false)** **M**

This Standard method prepares the "sound" device for subsequent **read**s and/or **write**s. An argument can be supplied (i.e., following a ':' in the path-name component, available via **my-args**) to specify sampling parameters. The argument is a string consisting of the external representation of the sample-rate to be used; if absent, an implementation-defined sample rate is used. An error is signalled by **open** (i.e., it returns **false**) if the requested sample rate can not be supported by the device.

close **(--)** **M**

Standard Open Firmware behavior.

read **(addr size -- actual)** **M**

Acquire sound data, storing the samples at addr.

write **(addr size -- actual)** **M**

Output sound samples, stored at addr.

7. NVRAM Device

Access to NVRAM is supported by this device-type. The NVRAM is treated as a device that can be read and written using the standard Open Firmware **read** and **write** methods; the starting position within the NVRAM can be specified by the **seek** method.

7.1. NVRAM properties

"device_type" **S**

Standard *property*: the value of this property *shall* be "nvram".

"#bytes" **S**

prop-name, describes the number of bytes the device is capable of storing.

prop-encoded array: An integer, encoded as with **encode-int**. The value is the number of bytes the device can store. Typical values could be 4K, 8K, 16K or 32K bytes.

7.2. NVRAM methods

The following methods have the semantics of the Open Firmware methods:

read **(addr len -- actual)** **M**

write **(addr len -- actual)** **M**

seek **(pos.lo pos.hi -- status)** **M**

size **(-- size)** **M**

The returned value, **size**, is the NVRAM size in bytes.

Note: Size should not include area that is pre-allocated by the platform; i.e., such as a VPD(Vital Product Data) area.

The following methods have additional behavior depending upon the *argument* used to open the device.

open **(-- true | false)** **M**

Standard method used to initiate access to the device

close **(--)** **M**

Standard Open Firmware behavior

8. Parallel port device

Access to the parallel port is supported by this device-type. The parallel port is treated as a byte-stream device.

8.1. Parallel port properties

"device_type" **S**

Standard *property*; the value of this property *shall* be "parallel".

8.2. Parallel port methods

The following methods have the semantics of the corresponding Open Firmware standard methods:

open **(-- true | false)** **M**

close **(--)** **M**

write **(addr len -- actual)** **M**

9. Additional Requirements for SCSI Devices

A node representing a SCSI device *shall* implement all the methods and properties specified in Annex E, sections E.1 through E.5 of [1].

9.1. Properties for SCSI Bus Nodes

"wide" **S**

prop-name, indicates that the SCSI Node is wide.

prop-encode-array: <none>

This property *shall* be present if the SCSI controller represented by this node is wide, with SICSI id ranges 0 through 15, and *shall* be absent otherwise.

In order to support the wide SCSI, the example in Appendix E.6.3 of [1] would be amended as follows:

```
E.6.3hacom.fth
...
: show-children  ( -- )
  open  0=  if  ." Can't open SCSI host adapter" cr  exit  then
" wide"  get-my-property  if  8  else  2drop d# 16  then
  0 do
     i probe-target  if
        ." Target " i . cr
        8 0  do  i j show-lun  loop
     then
  loop
;
```

"differential"

prop-name, indicates that the SCSI Node supports differential signaling.

prop-encode-array: <none>

This property *shall* be present if the SCSI controller represented by this node supports differential signaling.

"scsi-initiator-id"

prop-name, indicating the initiator-id to be used for SCSI transfers by this controller.

prop-encode-array: An integer, encode as with **encode-int,** that specifies the value of the initiator-id for subsequent transfers by this controller. If the "wide" property is not present the value is in the range of

0..7. If the "wide" property is present the value is in the range of 0..15. The initial value of the property is implementation dependent.

9.2. Methods for SCSI Bus Nodes

open (-- true) M

In addition to the standard Open Firmware behavior, the open method *shall* set the host adapter's own selection ID as follows:

Attempt to locate a "scsi-initiator-id" property by executing "get-inherited-property" with the string "scsi-initiator-id" as its argument. If such a property is found, decode its value as with "decode-int", and use the decoded value as the host adapter's own selection ID. Otherwise, use the value 7.

Note: The use of "get-inherited-property" to get the "scsi-initiator-id" property makes it possible to choose the set of SCSI host adapters to which the property applies. If the property is in the root node, for example, it applies to all the SCSI host adapters in the system. If the property is elsewhere in the device tree, it applies only to host adapters in the subtree below and including the location of the property.

10. Additional Requirements for Block and Byte devices

The "disk-label' standard support package and packages of device type "block" and "byte" *shall* implement the following method:

size (-- d) M

Return the size of the device in bytes.

Return, as a double number "d", the number of bytes of storage associated with the device or instance. If the size cannot be determined, return the double number -1.

Packages of device type "block" and "byte" *shall* implement the following method:

#blocks (-- u) M

Return the size of the device in blocks.

Return, as an unsigned number "u", the number of blocks of storage associated with the device or instance, where a "block" is a unit of storage consisting of the number of bytes returned by the package's "block-size" method. If the size cannot be determined, or if the number of blocks exceeds the range of an unsigned number, return the maximum unsigned integer (which, because of Open Firmware's assumption of two's complement arithmetic is equivalent to the signed number -1).

11. Additional Requirements for Block Devices

The "disk-label" standard support package and packages of device type "block" *shall* implement the following methods:

offset-low (-- u) M

Returns the least significant cell of the double number denoting the beginning offset of the disk partition that was specified when the "disk-label" support package was opened. In general that offset is obtained by executing the **offset** method of the "disk-label" support package with an agrument of zero. It is permissable for the disk package to execute the "disk-label" support package's **offset** method once after opening that support package, storing the result for later use.

offset-high (-- u) M

Returns the most significant cell of the double number denoting the beginning offset of the disk partition that was specified when the "disk-label" support package was opened. In general that offset is obtained by executing the **offset** method of the "disk-label" support package with an agrument of zero. It is permissable for the disk package to execute the "disk-label" support package's **offset** method once after opening that support package, storing the result for later use.

12. /aliases Node Properties

The following aliases *shall* be provided under the "/aliases" node if an applicable device node exists:

keyboard

mouse

rtc

sound

nvram

parallel

- END OF DOCUMENT -