Open Firmware
Recommended Practice:

Device Support Extensions

Version 0.8 [Interim Version]

August 21, 1996 3:02 pm

Unapproved Draft

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/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

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 verbiage. 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.
Revision 0.6	03/11/96	Changed NVRAM Note in Section 7.2 and the differential property definition in Section 9.1. Made several editorial changes.
Revision 0.7	08/06/96	Changed document format from binding to recommended practice. Removed keyboard language-specific layout section and device methods scan-codes. Added Serial Device Section 8.0. Changed SCSI to SCSI-2, Section 10.0.
Revision 0.8	08/19/96	Moved Section 3.3, keyboard 'abort key sequence', to keyboard/mouse binding. Included proposal #350 to rewrite the `"sound"` Section, Section 6.0. Removed all properties except the **`"device_type"`** for the `"serial"` device section 8.0. Added new Section 9.0, `"network"` device.

1. Introduction

This recommended practice 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.

1.1. Purpose

This recommended practice 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 should 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.

1.2. Scope

New platform bindings are encouraged to require compliance with this recommended practice. New implementations, based upon existing requirements and/or an architecture, may choose not to comply with this document to prevent compatibility problems for the devices described herein.

2. References and Definitions

2.1. 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.*

[4] *Open Firmware: Recommended Practice - Generic Names.*

[5] *Open Firmware: Recommended Practice - Interrupt Mapping.*

[6] *ISA/EISA/ISA-PnP binding to: IEEE Std. 1275-1994 Standard for Boot (Initialization, Configuration) Firmwar*e.

2.2. Definitions

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 remap 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.

3.1. "keyboard" device standard properties

`"device_type"` **`S`**

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

`"language"` **`S`**

prop-name, that indicates the current scan-code to character conversion.

prop-encoded-string: a string, as encoded with **`encode-string`**.

The string 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

Keyboard devices *shall* implement **`open`**, **`close`** and **`read`** methods as defined by the Open Firmware standard[1]. Note that the **open** routine can take an optional argument which specifies the language (i.e., scancode to character mapping) to be used.

3.3. 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.

Sequence	Key legend
CSI[a] A	up arrow
CSI B	down arrow
CSI C	right arrow
CSI D	left arrow
CSI H	home
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 @	insert
CSI P	delete[b]
CSI K	end
CSI ?	page-up
CSI /	page-down
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. Note: 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**

prop-name, that indicates the number of physical buttons supported by the device.

prop-encoded-int: an integer, as encoded with **encode-int**.

The property value is an integer that indicates the number of buttons for the pointing device. This property can be used to interpret the *buttons* value returned by the **get-event** method.

"absolute-position" **S**

prop-name, that indicates the device supplies absolute **X**,**Y** coordinates.

prop-encoded-array: <none>

The presence of this property indicates that the 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 method *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 (msec -- pos.x pos.y buttons true|false)M

This method is used to obtain the next *event* of the pointing device. msecs is the number of milliseconds to wait for an event before reporting failure; a value of zero (0) implies wait until event pos.x, pos.y return the positioning information. The pos.x and pos.y values can be interpreted as unsigned or signed depending upon the presence or absence of the **"absolute-position"** property. The value for pos.x increases

moving to the right direction and the value for `pos.y` increases moving in a downward direction. `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 top stack result indicates whether an event was detected within the timeout 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, accept audio to write or return audio from read that consists of a sample rate of 8KHz 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 anticipated 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 capabilities, so long as a default `open` results in `read` and `write` using the standard format.

6.1. Definition of Terms

channel: One "track" of audio data. In stereophonic data, there are two channels - left and right.

sample: The atomic unit of audio data. A single digital sample from one channel.

sample frame: A set of samples, one per channel. (In the trivial case of a single channel, synonymous with *sample*.)

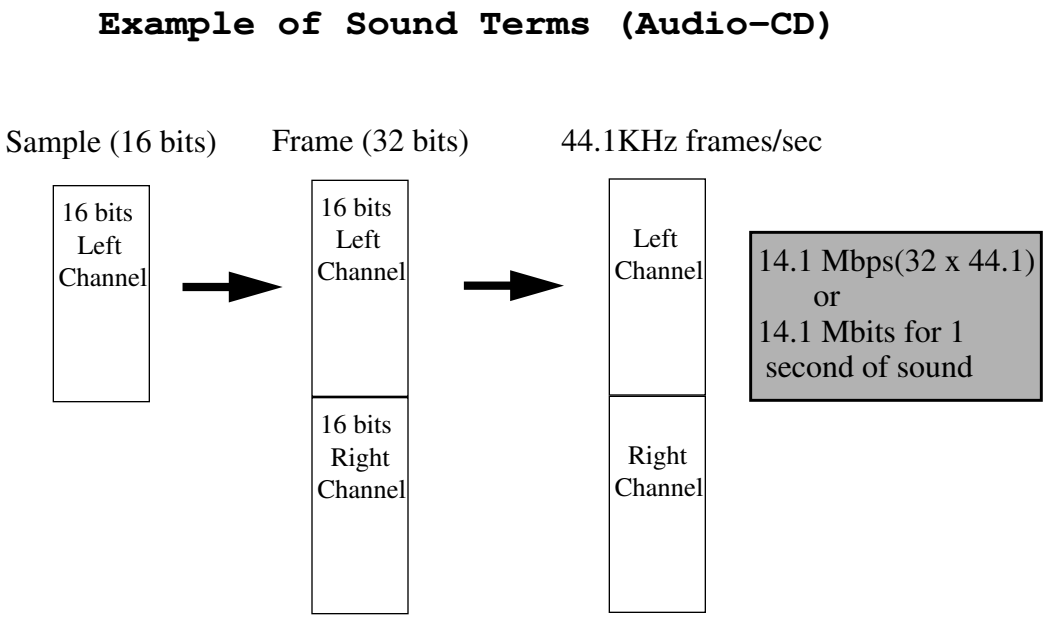
precision: The size (in bits) of a sample.

signed linear: An audio encoding where the sample is represented by a signed number, with 0 as the centerpoint.

unsigned linear: An audio encoding where the sample is represented by an unsigned number with 1/2 range, e.g. 128 or 32768, as the centerpoint.

6.1.1. Example of sound terms (Standard Audio-CD)

A standard audio-CD player has a digital output sound channel that uses unsigned linear unencoded (PCM) 16 bit sample precision per channel (2 channels for stereo) at a frame rate of 44.1KHz. The sound information would be defined in the above terms by:



6.1. "sound" device standard properties

"device_type" **S**

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

"#input-channels" **S**

Prop-name, defines the possible numbers of input channels supported.

prop-encoded-array: Arbitrary number of integers, each encoded with **encode-int**.

Each integer specified a possible number of input channels. "1" indicates that the device is capable of monophonic operation. "2" indicates that it is capable of stereophonic operation. A device capable of both modes would include both values.

"#output-channels" **S**

Prop-name to define the possible numbers of output channels supported.

prop-encoded-array: Arbitrary number of integers, each encoded with **encode-int**.

Each integer specified a possible number of output channels. `"1"` indicates that the device is capable of monophonic operation. `"2"` indicates that it is capable of stereophonic operation. A device capable of both modes would include both values.

"sample-precisions" **S**

Prop-name to define the possible sample precisions.

prop-encoded-array: Arbitrary number of integers, each encoded with **encode-int**.

Specifies the possible numbers of bits required to store one audio sample from one channel.

"sample-frame-size" **S**

Prop-name to define the possible sample frame sizes.

prop-encoded-array: Arbitrary number of integers, each encoded with **encode-int**.

Specifies the possible numbers of bits required to store one sample frame - one sample from each channel.

"input-frame-rates" **S**

Prop-name to define the possible sample frame rates for audio input.

prop-encoded-array: Arbitrary number of integers, each encoded with **encode-int**.

Specifies the possible input sampling rates, in sample frames per second.

"output-frame-rates" **S**

Prop-name to define the possible sample frame rates for audio output.

prop-encoded-array: Arbitrary number of integers, each encoded with **encode-int**.

Specifies the possible output sampling rates, in sample frames per second.

"input-encoding-types" **S**

Prop-name to define the possible input encoding types.

prop-encoded-array: The concatenation, with **encode+**, of an arbitrary number of text strings, each encoded with **encode-string**.

Specifies possible input encodings. Valid values could include:

```
"8bit-u-law"
"8bit-A-law"
"8bit-unsigned-linear"
"8bit-signed-linear"
"16bit-LE-unsigned-linear"
"16bit-BE-unsigned-linear"
"16bit-LE-signed-linear"
"16bit-BE-signed-linear"
"4bit-ADPCM"
```

"output-encoding-types" **S**

Prop-name to define the possible output encoding types.

prop-encoded-array: The concatenation, with **encode+**, of an arbitrary number of text strings, each encoded with **encode-string**.

Specifies possible output encodings. Valid values are as for **"input-encoding-types"**.

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 reads and/or writes.

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 returned by the `size` method is 0 to *n*-1 where *n* is the size defined by the platform.

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. Serial port device

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

8.1. Serial port properties

As specified in [1] and [6] , with the following additions or modifications.

`"device_type"` **S**

Standard *property name* to define the device's implemented interface.

The meaning of this property is as defined in the Open Firmware core document [1]. The value for nodes described by this specification *shall* be "`serial`".

8.2. Serial port methods

As specified in [1] and [6], with the following additions or modifications.

8.2.1. Device Arguments for `Open` Method

`open` is a standard method for the serial device. When the serial device is opened, several device-arguments can be passed as defined below:

driver-name@*unit-address*:***device-arguments***

device-arguments are defined to be of the form, e.g. 9600,8,n,1, and sets the UART parameters (baudrate, #data bits, parity, #stop bits, and handshake) accordingly.

The fields are (in order, left to right):

`<baud rate>, <data bits>, <parity>, <stop bits>, <handshake>`

Fields with empty values are not changed. Values for fields are whatever the hardware will support:

`baud rate`: various, including 110, 300, 1200, 2400, 4800, 9600, 19200, 38400

`data bits`: 5,6,7,8

`parity`:

char	means
n	none
e	even
o	odd
m	mark
s	space

`stop bits`:

char	means
1	1 stop bit
.	1.5 stop bits
2	2 stop bits

`handshake`:

char	means
-	none
h	hardware (rts/cts)
s	software (xon/xoff)

Note: The serial device default mode is 9600,8,n,1,- if the `open` arguments are not specified.

8.3. Device-specific Methods for Device Nodes

As specified in [1] and [6], with the following additions.

`set-mode` `( adr len -- )` **M**

Sets the device mode according to the string at *adr*, of length *len*, which is interpreted the same as the arguments to the **`open`** method (see section 8.2.1.1. on page 9).

`set-modem-control` `( bitmask -- )` **M**

Sets the device's modem control lines according to the following table:

bitmask	Modem control state
0	RTS off, DTR off
1	RTS off, DTR on

bitmask	Modem control state
2	RTS on, DTR off
3	RTS on, DTR on

The response for other values of *bitmask* are implementation dependent.

Changing the bits associated with these lines *shall* have no effect on other bits not associated with this function.

9. "network" device

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

9.1. "network" device properties

These properties *shall* be reported by packages representing **"network"** devices.

"device_type" **S**

Standard *property*: the value of this property for network devices *shall* be "network".

"supported-network-types" **S**

prop-name, reports possible types of "network" the device can support.

prop-encoded-array: a string, as encoded with **encode-sting**.

The string is chosen from the following set representing <network type>, <speed (Mbps)>, <connector type> and <duplex mode>:

```
"ethernet,10,rj45,half"
"ethernet,10,rj45,full"
"ethernet,100,rj45,half"
"ethernet,100,rj45,full"
"ethernet,10,aui,half"
"ethernet,10,aui,full"
"ethernet,100,aui,half"
"ethernet,100,aui,full"
"ethernet,10,bnc,half"
"ethernet,10,bnc,full"
"ethernet,100,bnc,half"
"ethernet,100,bnc,full"

"token-ring,4,rj45,half"
"token-ring,4,rj45,full"
"token-ring,16,rj45,half"
"token-ring,16,rj45,full"
"token-ring,4,9pin,half"
"token-ring,4,9pin,full"
"token-ring,16,9pin,half"
```

```
                "token-ring,16,9pin,full"

                "fddi,100,rj45,half"
                "fddi,100,sc,half"
                "fddi,100,mic,half"

                "atm,100,sc,full"
                "atm,155,sc,full"
                "atm,622,sc,full"

                "fcs,1000,sc,full"
"chosen-network-type"                                          S
```

prop-name, reports type of "network" this device is supporting.

prop-encoded-array: a string, as encoded with **encode-string,** that is chosen from one of the values in **"supported-network-types"** property.

If present, indicates that the firmware or the user has selected one of the **"supported-network-types"** and the value indicates which one was chosen. The **"chosen-network-type"** property need not exist if the firmware cannot determine the "network" type or there is only a single "network" type choice.

9.2. "network" device methods

As specified in [1], Section 3.7.4, the standard methods are supported with the addition of new arguments passed through the **open** method. All arguments are defined to be optional. The arguments defined for the "network" device are parsed by the **open** method from the input string of arguments and are not passed through; i.e., to the "TFTP" package, as an example. Values for device arguments are defined below:

open *network-device*:[promiscuous,][speed=*n*,][duplex=*mode*,]

where:

promiscuous	puts network device in a mode where all addresses are recognized; e.g., a communication device that *snoops* the LAN.
speed=*n*	where *n* indicates the "network" speed in Mbps. Typical values can be 4, 10, 16, 100, 155, 622 & 1000 Mbps.
duplex=*mode*	where *mode* values are "half" or "full".

If the **open** method does not recognize a value for *n* or *mode* parameters or the "network" device cannot execute the specified value for *n* or *mode* (speed or duplex mode), a failure should result (device should not open) with a warning message to the user.

10. Parallel port device

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

10.1. Parallel port properties

"device_type" **S**

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

10.2. 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**

11. Additional Requirements for SCSI-2 Devices

A node representing a SCSI-2 device *shall* implement all the methods and properties specified in Annex E, sections E.1 through E.5 of [1].

11.1. Properties for SCSI-2 Bus Nodes

"wide" **S**

prop-name, indicates that the SCSI-2 Node is wide.

prop-encode-array: <none>

This property *shall* be present if the SCSI-2 controller represented by this node is wide, with SCSI-2 ID ranges 0 through 15, and *shall* be absent otherwise.

In order to support the wide SCSI-2, 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-2 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-2 Node supports differential signaling.

prop-encode-array: <none>

This property may be present if the SCSI-2 controller represented by this node supports differential signaling.

"scsi-initiator-id"

prop-name, indicating the initiator-id to be used for SCSI-2 transfers by this controller.

prop-encode-array: An integer, encoded as with **encode-int**.

The integer 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.

11.2. Methods for SCSI-2 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-2 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.

12. 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).

13. 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 argument of zero. It is permissible 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 argument of zero. It is permissible 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.

14. /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

serial

network

-- END OF DOCUMENT --