PowerPC™ Microprocessor

Common Hardware Reference Platform (CHRP™)

System binding to:

IEEE Std 1275-1994

Standard for Boot

(Initialization, Configuration)

Firmware

Revision: 1.5 (UNAPPROVED DRAFT)

Date: May 8, 1996

Purpose of this PowerPC Microprocessor CHRP™ System binding

This document specifies the application of Open Firmware to a PowerPC Microprocessor CHRP System, including requirements and practices to support unique hardware and firmware specific to the platform implementation. The core requirements and practices specified by Open Firmware must be augmented by system-specific requirements to form a complete specification for the firmware implementation of a PowerPC Microprocessor CHRP System. This document establishes such additional requirements pertaining to the platform and the support required by Open Firmware.

Task Group Members

The following individuals were members of the Task Group that produced this document:

Rob Baxter, Motorola®

Mitch Bradley, FirmWorks

Steve Bunch, Motorola

Bob Coffin, IBM® (editor)

Ron Hochsprung, Apple®

John Kingman, IBM

Dr. Luan Nguyen, IBM

John O'Quin, IBM

Andy Rawson, IBM

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 or trademarks of the companies as shown in the list below:

Trademark	Company
AIX™	International Business Machines Corporation
Apple	Apple Computer, Inc.
CHRP	Apple Computer, Inc.
Ethernet™	Xerox
IBM	International Business Machines Corporation
Microsoft®	Microsoft Corporation
Macintosh™	Apple Computer, Inc.

Mac™ OS	Apple Computer, Inc.
Motorola	Motorola, Inc.
PowerPC	International Business Machines Corporation
Windows NT™	Microsoft Corporation

Revision History

Revision 0.1	07/27/95	Initial draft; limited distribution to CHRP AIM(Apple, IBM & Motorola) Team
Revision 0.2	08/15/95	Initial draft with distribution to CHRP AIM Team for review.
Revision 0.3	09/08/95	Distribution to CHRP AIM Team for review. Added more function to Open Firmware(properties & methods) and RTAS.
Revision 0.4	10/20/95	Distribution to CHRP AIM Team for review. Changed basic format and structure of document with appendices for proposals. Added function/expanded function to: 'bootp', Power Management, System Nodes. Expanded review process to include FIrePower, FirmWorks and SunSoft.
Revision 0.5	11/11/95	Added description material to Appendix A for (proposed)interrupt support. Expanded review process to include Bull, Canon & Toshiba.
Revision 0.6	12/07/95	Added Appendix A for Future Direction. Differed certain function to next version of CHRP System binding. CHanged old Appendix A to Appendix B for an interrupt proposal as a recommended practice format. Numerous editorial changes.
Revision 0.7	02/07/96	Removed Appendix A; an example of CHRP Device Tree. Appendix A became Interrupt Structure. Changed multi-boot in Section 3.0. Moved Sections of PowerPC Processor binding Document to system binding; ELF, endianess, properties, Client Callbacks. Numerous editorial changes. Internal IBM Version.
Revision 0.8	02/08/96	Minor changes to version 0.7 for review with AIM Members in Tempe, AZ.
Revision 0.9	02/23/96	Changed Multi-boot and ELF sections to clarify. Made numerous editorial changes for accuracy and clarification.
Revision 1.0	04/10/96	Changes to reflect AIM working Cupertino Meeting on 03/7&8/96. Added references(CHRP I/O device bindings) and edited references. Edited boot flow(Section 3) to add path names and SGML tag format for bootinfo. Combined ROM Node(5.2.1.1) to include OS-ROM function. Numerous editorial changes to ISA and "disk-label" Support Package Sections.
Revision 1.1	04/16/96	Made changes for AIM Meeting, 04/15/96 in Tempe, AZ. Edited references 15, 21, 23 thru 32. Edited Boot Overview and the **`/options`** Node. Edited Table 1 end eliminated Note 2. Changed the tags in the 'bootinfo.txt' file. Redefined the ICON string definition. Changed Boot Script Description to **`reboot-command`** Description. Abbreviated Section 5.7.1.1 by referencing [14]. Edited the ISA Device Support Section 8.0. Added Load Method,

		Section 10.1.3 and modified **"tape"** package(10.2).
Revision 1.2	04/22/96	Made changes from AIM Meeting, 04/16/96 in Tempe, Az. Changed 'boot Overview(Section 3.0) and made 'bootinfo.txt' format use SGML Tags. Added **reboot-command**. Added material to ISA Device Support (Section 8.0). Eliminated Mac OS file support for the **"disk-label"** package. Added/changed material to OF NVRAM SECTION 11.0.
Revision 1.3	04/29/96	Removed PE Section 9.4.2. Made changes to CHRP Boot Flow Section 3.0. Changed 'bootinfo.txt' tags in Section 3.1.3. Changed System Control Area section 5.2.1 and Removed 5.2.1.2. Modified Section 12.0, OF NVRAM to change partition definition. Changed Event Sources Node Section 5.9.1. Changed ISA Device Support Section 8.0. Changed NVRAM Partition Section 12.0.
Revision 1.4	05/02/96	Numerous editorial changes regarding Open Firmware practices about names of properties and methods. Corrected the Table of Contents and added a List of Tables. Added/changed material in Section 9.1, ISA Configuration Utility (ICU). Modified NVRAM Partition Section 12.0.
Revision 1.5	05/08/96	Numerous editorial changes regarding document to make consistent with Open Firmware convention and improve readability. Changed Section 3.0 to have a boot overview. Renamed most of section to 'Multi-boot'. Made changes to **root** node Section 5. Changed Section 12.0; modified OS naming convention and classified System Partition Configuration Variables as to category(boolean, string or number). Rewrote all of Section 12.4, System Partition.

Table of Contents:

List of Tables:

1. Overview

This document specifies the application of *IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware, Core Practices and Requirements*, *Core Errata, IEEE P1275.7* and appropriate Open Firmware Standards for PowerPC Microprocessor CHRP Computer Systems[3], including practices for client program interface and data formats.

1.1. General Requirements for Open Firmware

An Open Firmware implementation for a PowerPC Microprocessor CHRP platform *shall* implement the core requirements as defined in [1], core errata[2], the PowerPC Processor-specific extensions described in [5], other appropriate bindings and/or recommended practices contained in the references (Section 2.1), and the PowerPC Microprocessor CHRP binding specific extensions described in this document.

In addition, an Open Firmware implementation for a PowerPC Microprocessor CHRP platform *shall* implement the *Device Interface*, *Client Interface* and *User Interface* as defined in Open Firmware core document, reference [1].

The Open Firmware Documentation can be found on server, '**playground.sun.com.**'. The URL for this documentation is the following:

http://playground.sun.com/1275

or

ftp://playground.sun.com/1275

Note: The Open Firmware Documentation at the above site is mirrored at http://chrp.apple.com/1275

Versions of the Open Firmware Documentation are maintained in sub-directories under the *1275* directory on a topic basis. For example, under the *bindings/* sub-directory, the *isa/*, *pci/* and *ppc/* sub-directories have versions of the *ISA/EISA/ISA-PnP binding*, *PCI Bus binding* and *PowerPC Processor binding* respectively. The sub-directory *postscript/* contains the latest postscript versions of the Open Firmware Documentation.

For ISA support, the CHRP System binding has additional restrictions and/or requirements imposed on the *ISA/EISA/ISA-PnP binding*(See "Additional ISA Requirements" on page 42.). Also, for completeness, the ISA I/O Device bindings([22],[23],[24],[25],[25],[26],[27],[28],[29]&[30]) and the Apple/MAC I/O Device binding([31]) are listed in the references. The I/O device bindings support the CHRP Standard I/O specified in the I/O Device Reference Document[4].

2. References and Terms

References and terms for the CHRP System binding are listed in this section.

2.1. References

This Open Firmware System binding standard *shall* be used in conjunction with the following publications. When the following standards are superseded by an approved revision, the revision *shall* apply.

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

[2] *Core Errata, IEEE P1275.7/D4.*

[3] *PowerPC Microprocessor Common Hardware Reference Platform: A System Architecture*. This document describes the platform specifications; hardware attributes, characteristics and I/O Sub-system content.

[4] *PowerPC Microprocessor Common Hardware Reference Platform: I/O Device Reference*. This document describes the platform Standard I/O Devices; hardware registers, register locations, and hardware attributes.

[5] *PowerPC Processor binding to: IEEE Std 1275-1994, Standard for Boot (Initialization, Configuration) Firmware*.

[6] *PCI Bus binding to: IEEE Std 1275-1994, Standard for Boot (Initialization, Configuration) Firmware*.

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

[8] *PC Card binding to: IEEE Std 1275-1994, Standard for Boot (Initialization, Configuration) Firmware*.

[9] *Device Support Extensions: to IEEE 1275-1994 Standard for Boot (Initialization, Configuration) Firmware*.

[10] *Open Firmware: Recommended Practice - 16-color Text Extensions*.

[11] *Open Firmware: Recommended Practice - 8-bit Graphics Extensions*.

[12] *Open Firmware: Recommended Practice - Forth Source and FCode Image Support*. This document describes program-image formats for FCode and Forth Source Code.

[13] *Open Firmware: Recommended Practice - Generic Names*. This document describes a naming convention for Open Firmware device nodes.

[14] *Open Firmware: Recommended Practice - Interrupt Mapping*. This document describes an interrupt structure for Open Firmware.

[15] *Open Firmware: Recommended Practice - obp-tftp Extension.* This document describes network extensions to the OBP-TFTP Support Package.

[16] *MS-DOS Programmer's Reference*, published by Microsoft. This document describes the MS-DOS partition, directory and FAT formats used by disk-label support package.

[17] *Peering Inside the PE: A Tour of the Win32 Portable Executable File Format*, found in the March, 1994 issue of *Microsoft Systems Journal*.

[18] *ISO-9660, Information processing -- Volume and file structure of CD-ROM for information interchange*, published by International Organization for Standardization.

[19] *System V Application Binary Interface, PowerPC Processor Supplement*, Sunsoft. This document defines the PowerPC ELF format.

[20] Inside Macintosh, SCSI Disk Partition Definition, *CD-ROM ISBN 0-201-94591-6*.

[21] *Open PIC Multiprocessor Interrupt Controller Register Interface Specification*, Revision 1.2. Refer to URL of 'http://www.amd.com/html/products/pcd/openpic/19725c.pdf' for the document.

[22] *CHRP ISA Keyboard/Mouse Controller Device binding*.

[23] *CHRP ISA DMA Controller Device binding*.

[24] *CHRP ISA Parallel Port Device binding*.

[25] *CHRP ISA Audio Device binding*.

[26] *CHRP ISA Interrupt Controller Device binding*.

[27] *CHRP ISA Serial Port Device binding*.

[28] *CHRP ISA Floppy Controller Device binding*.

[29] *CHRP Linear Frame Buffer Display Device binding*.

[30] *CHRP VGA Display Device binding*.

[31] *MAC I/O binding*.

[32] *ISO Standard 8879:1986, Information Processing -- Text and Office Systems -- Standard Generalized Markup Language(SGML).*

[33] *Plug and Play ISA Specification*, Version 1.0a, 05/05/1994.

[34] *Clarification to the Plug and Play ISA Specification,* Version 1.0a, 12/10/94.

2.2. Terms

This standard uses technical terms as they are defined in the documents cited in "References", plus the following terms:

ARP: Address Resolution Protocol

BOOTP: Bootstrap Protocol

CHRP: Common Hardware Reference Platform, also PowerPC Platform.

core, core specification, core document: Refers to *IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware, Core Practices and Requirements*

core errata: Refers to *Core Errata*, *IEEE P1275.7*

CPU: Central Processing Unit

ELF: Executable and Linking Format. A binary object file format defined by [19]that is used to represent *client programs* in Open Firmware for PowerPC.

FDISK: Refers to the boot-record and partition table format used by MS-DOS, as defined in [16].

gateway: Network connecting device

host: A computer. In particular a source or destination of messages from the point of view of the communication network.

ICMP: Internet Control Message Protocol

ICU: ISA Configuration Utility; Refers to CHRP platform program to assist a user to configure ISA resources.

IETF: Internet Engineering Task Force

IP: Internet Protocol

IO: Input/Output

LAN: Local Area Network

NVRAM: Non-volatile memory that is the repository for various platform, Open Firmware and operating system information that remains persistent across reboots, power management activities and/or cycles.

Open Firmware: The firmware architecture defined by the core specification[1] and errata[2], or, when used as an adjective, a software component compliant with the core specification and errata.

Open PIC: Is a register-level architectural definition of an interrupt controller that supports up to 32 processors(Refer to [21] for more information).

PCU: Power Configuration Utility; Refers to CHRP platform program to assist a user to manage device power.

PE: Portable Executable. A binary object file format defined by [17]; this format is used by Microsoft's NT operating system.

PROM: programmable read only memory

real-mode: The mode in which Open Firmware and its client are running with translation disabled; all addresses passed between the client and Open Firmware are real (i.e., hardware) addresses.

RFC: Internet Request For Comments; part of the technical process of establishing a standard.

ROM: Read Only Memory

suspend: A form of Power Management characterized by a fast recovery to full operation. Typically, system memory will not be powered off while in the suspend state.

TFTP: Trivial File Transfer Protocol

UDP: User Datagram Protocol

virtual-mode: The mode in which Open Firmware and its client share a single virtual address space, and address translation is enabled; all addresses passed between the client and Open Firmware are virtual (translated) addresses.

3. CHRP Boot Flow

This section gives a system boot process overview and defines the enhancements to the standard Open Firmware boot process that are present in the boot process for a CHRP platform.

3.1. CHRP Boot Overview

The CHRP platform performs a normal Open Firmware boot(See [1], *IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware, Core Practices and Requirements*) as stated in the Core Practice Document, Section 4.2.3, Start-up script evaluation. CHRP platforms provide an additional capability to assist the user in choosing which of several operating systems to boot. A key sequence can be used to interrupt the normal boot flow and present the user with a *multiboot menu*, which can be either graphical or text-based at the discretion of the platform's firmware, from which the user can choose one of the installed or installable operating systems. Presenting the user with this choice can also be made the default mode of operation at platform boot time, by means of the **`auto-boot?`** and **`menu?`** configuration variables.

An overview of a CHRP platform boot sequence and the additions of the *multiboot menu* are given below:

CHRP Start-up Script Sequence Evaluation:

a) Power On Self Test(POST)

b) System Initialization

c) Evaluate the *script* (if **`use-nvramrc?`** is true)

d) **`probe-all`** (evaluate FCode)

e) **`install-console`**

f) **`banner`**

(key chord for *multiboot menu* will be recognized and acted upon after this point)

g) Secondary Diagnostics and other system-dependent initialization

h) Default boot (if **`auto-boot?`** is true)

i) Entry to *multiboot menu* (if **`menu?`** is true)

j) Invoke the command interpreter (if preceding step returns)

The CHRP boot flow described above occurs after all of the devices have been probed (i.e., by the execution of **probe-all**); See section 3.1.1. additional requirements for **probe-all** method.

The CHRP boot sequence defaults to a normal boot if the boolean variable **auto-boot?** is true and **diagnostic-mode?** is false. In this situation, the system *shall* then boot from information contained in the configuration variables **boot-device** and **boot-file**.

From the boot sequence above, entry to the *multiboot menu* may occur anywhere after step 'f', **banner**, if the platform key sequence(*multiboot menu*) has been depressed or in step 'i' if the boolean variable **menu?** is true.

3.1.1. Additional Requirements for probe-all Method

Before probing for plug-in devices, Open Firmware *shall* execute the **probe** method, as with **execute-device-method**, of any built-in device nodes. The order of evaluation *shall* ensure that the **probe** method of a parent device node is executed before the **probe** method of any of its children.

> **Note: During this built-in probing, /rom nodes will locate ROM based OSs. The FCode for these devices can publish their "bootinfo" properties that are used during the multiboot scenario as described below.**

3.1.2. CHRP Multiboot

The boot choices identified to the user are defined by *bootinfo objects* which are located on various system media. Each *bootinfo object* contains information about one operating system, such as its name and description, an icon depicting it, and an Open Firmware command sequence to load and execute it. The locations where *bootinfo objects* can be found are specified by Open Firmware *device-specifiers* that are the values of configuration variables, the names of which are of the form **"bootinfo-nnnnn"**, where **"nnnnn"** is OS-specific. These *configuration variables* are stored in the System Partition in NVRAM and are published in the *device tree* as properties under the **/options** node. The *multiboot menu* will use these *configuration variables* to locate and parse *bootinfo* to obtain the OS icon, description, etc.

In addition to the **bootinfo-nnnnn** configuration variables, the *multiboot menu* will search the device tree for nodes containing **"bootinfo"** properties, which specify that the node can supply a *bootinfo object*. This is particularly useful for operating systems contained in ROMs.

> **Note: The order prescribed by probe-all guarantees that these properties be created before the *multiboot menu* has been invoked.**

Different versions of the same operating system may each have their own *bootinfo* and associated *configuration variables*. Although it is possible to put *bootinfo* in any media location that Open Firmware can read, this specification defines standard locations for various types of media, to allow the firmware to establish the *bootinfo configuration variables* automatically in many cases.

3.1.3. Bootinfo Configuration Variables

A *bootinfo configuration variable* is any *configuration variable* that meets the following requirements:

- Its name is of the form **"bootinfo-nnnnn"**, where *nnnnn* is a string of at most 22 characters from the set of valid characters for Open Firmware configuration variable names. The exact value of **"nnnnn"** for a particular operating system may be chosen by that operating system. The naming convention for the operating system should be chosen to avoid possible naming conflicts between operating system vendors(See Section 12.4.1. on page 64.)
- Its value is an Open Firmware *device-specifier* that identifies an object (e.g. disk file, tape file, disk partition or **/rom** child node) whose contents are a *"bootinfo object"* as defined below.

3.1.4. Bootinfo Properties

Any node in the device tree can have a **"bootinfo"** property whose value specifies the arguments to use in opening that device in order to access its *bootinfo object*.

"bootinfo" **S**

prop-name, locates the node's *bootinfo object*

prop-encoded-array: A string, encoded as with **encode-string**

The presence of this property signifies that the device has an associated *bootinfo object*. The value is a text string such that when this devices node **open** method is called, the value of text string that is passed to the device's node **open** method is **my-args**. When so opened, subsequent calls to the node's **"read"** method will yield the contents of the node's *bootinfo object*.

3.1.5. Standard Locations for *Bootinfo Objects*

The standard locations for *bootinfo objects* on various CHRP media and partition types is shown in the table below. An operating system must put it's *bootinfo object* in the standard location in order to guarantee interoperability with the CHRP *multiboot menu* mechanism.

Table 1. Standard Pathnames for *bootinfo.txt* file

Name	Device/Partition	Notes
OS Execution Media:		
Raw Partition:(0x41, Mac OS)	*device*:*partition #*	Entire contents of partition is bootinfo.txt
Any File System:	*device*:*partition*,*directory*\bootinfo.txt	Contents of file
ROM:	*device*:bootinfo	bootinfo is the value of the **"bootinfo"** property in a **/rom** child node
Tape:	Presence of bootinfo.txt is optional	Contents of file or part of contents of partition is bootinfo.txt
Network:	Could specify bootinfo.txt or some other file from the Bootp server	Specifying bootinfo.txt from the Bootp server is optional
Installation Media:		
Any block device:	*device*:*partition*,\ppc\bootinfo.txt	Any file system format
Tape:	*device*:0	

Example of install(**`"bootinfo-nnnnn"`**) block device(disk):

ALIAS EXAMPLE:

`bootinfo-aix-4.3=disk:2` (The contents of partition 2, which is probably a "0x41" partition, on the default disk, is the *bootinfo.txt* file for a version of the AIX Operating System.)

`bootinfo-nt-4.0=disk:\os\winnt\bootinfo.txt` (The file `\os\winnt\bootinfo.txt` on the default partition of the default disk is the *bootinfo.txt* file for a version of the Windows NT Operating System.)

NON-ALIAS EXAMPLE:

`bootinfo-aix-4.4=/pci@ff500000/pci3,1000@10/sd0,0:3` (The contents of partition 3, which is probably a "0x41" partition, on the SCSI disk at target 0 unit 0, is the *bootinfo.txt* file for a version of the AIX Operating System.)

3.1.6. Bootinfo Objects

The information used by Open Firmware to display information in the *multiboot menu* and to locate and process an OS load image is contained within a sequence of text that is called a *bootinfo object*. The text comprising the bootinfo object uses SGML[32] syntax, with tags identifying the subordinate elements.

The following outline is a summary of the organization of the *bootinfo object*. Elements at the same level do not have any required order. The tags are illustrated in upper case, but *shall* be processed in a case-insensitive manner.

```
<CHRP-BOOT>

   <DESCRIPTION>

      ....

   </DESCRIPTION>

   <OS-NAME>

      ....

   </OS-NAME>

   <BOOT-SCRIPT>

      ....

   </BOOT-SCRIPT>

   <ICON

      SIZE=ww,hh

      COLOR-SPACE=r,g,b

      >

         <BITMAP>

            hh hh hh hh ....

         </BITMAP>

   </ICON>

</CHRP-BOOT>
```

Note 1: If 'SIZE' is not present, assume default of 64,64. If 'COLOR-SPACE' is not present, assume default of 3,3,2.

Note 2: Another <chrp-boot> tag sequence could define a different boot selection

Note 3: CHRP platforms will recognize only the tags between the beginning <chrp-boot> tag until the end </chrp-boot> tag. If a tag is unrecognized, the material will be ignored until the end tag. Other non-<chrp-boot> tags may be supported in the future. These additional selections would also be presented to the user as boot options.

3.1.6.1. Bootinfo Entities

SGML provides "entities" that provide symbolic names for text. When the entity names are contained within & and ';', the entity is replaced with text as defined by the entity; i.e., entities provide a "macro" substitution capability. The *bootinfo object* may use entities to supply pathname components that depend upon the location of the file. Also, entities have been defined for the standard SMGL Tags for the presence of the '<', '&' and '>' characters in the text as `<` , `&` and `>`. Within the <BOOT-SCRIPT> element, the following entities are defined with respect to the fully qualified pathname of the *bootinfo object*:

device	the device component.
partition	the partition component.
directory	the directory component.
filename	the filename component.
full-path	the entire fully qualified pathname.

The fully qualified pathname could be represented by the following text:

&device;:[*&partition;*][,]*&directory;**&filename;*

Note: underlined portions illustrate where entities are positioned within the full pathname

3.1.6.2. Bootinfo Character Sets

The character set used by the bootinfo.txt file is ISO-8859-1 (Latin-1). Element tags and entity names are not case sensitive; all other text is case sensitive.

3.1.6.3. Element Tag Descriptions

The following sections describe each of the element tags and how they are used.

3.1.6.4. CHRP-BOOT Element

This element provides the grouping for each OS that is represented within a single `bootinfo.txt` file. Multiple CHRP-BOOT sections are allowed within a single `bootinfo.txt` file.

3.1.6.5. OS-NAME element

This element contains the complete name of the OS.

3.1.6.6. BOOT-SCRIPT element

This element contains an Open Firmware script that is executed when the OS defined by this CHRP-BOOT section is selected to be loaded. Each line of this element is processed as if it were entered from the input device of the user interface. Typically, the last line of this script would contain a **`boot`** command; the pathname of the OS's load image can be constructed with the entities described above.

3.1.6.7. ICON element

This element describes the OS icon that can be displayed by the multi-boot process. The icon should be designed to be pleasant against a light background.

The SIZE parameter consists of a two decimal numbers, separated by a comma, that represent the width and height (in pixels) of the icon, respectively. The default value is “64,64”

The COLOR-SPACE parameter consists of three decimal numbers, separated by commas, that represent the number of bits for the red, green, and blue components of each pixel. The default value is “3,3,2”[1].

> **Note 1: This version of CHRP supports only a 3,3,2 icon color-space and 64,64 icon size. Other icon size’s and color-space’s are reserved for future CHRP implementations**

If an icon is not stated, the platform will display a generic system icon that is platform dependent.

3.1.6.7.1. BITMAP element

This element specifies the bitmap. It consists of a sequence of hex digit pairs, each of which defines a pixel; white spaces is allowed between pixel values. The number of hex digit pairs is defined by the product of the width and height values of the SIZE parameter.

icon string example:

```
<icon size=64,64 color-space=3,3,2><bitmap>hh hh...
hh[2]</bitmap></icon>
```

> **Note 2: Hex string would be 8192 characters for a size=64,64 in the above example.**

For the two examples below, the tags have been indented and separated by line feeds for each start/end tag pair to make a more readable script style.

AIX Bootinfo Object Example:

```
<chrp-boot>
   <description>AIX 4.2.D.0</description>
   <os-name>AIX 4.2.D.0</os-name>
   <boot-script>boot &device;:2</boot-script>
   <icon size=64,64 color-space=3,3,2><bitmap>hh ... hh[1]</bitmap></icon>
</chrp-boot>
```

AIX Diagnostics Bootinfo Object Example:

```
<chrp-boot>
   <description>AIX 4.2.D.0 Diagnostics</description>
   <os-name>AIX 4.2.D.0 Diagnostics</os-name>
   <boot-script>boot &device;:2 diag</boot-script>
   <icon size=64,64 color-space=3,3,2><bitmap>hh ... hh[1]</bitmap></icon>
</chrp-boot>
```

> **Note 1: 64x64 icon size would have 8192 hex string characters**

3.1.7. Multiboot Menu

If the boot sequence is interrupted by the multiboot key sequence, then the firmware *shall* present a *multiboot menu* that provides at least the functions listed below. The form of the menu (e.g. graphical or text- oriented) and the selection mechanism (e.g. numbered choices, arrow keys, or mouse) are platform-dependent.

Multiboot Required Functions:

- Locate all bootinfo objects specified by bootinfo configuration variables and device node **"bootinfo"** properties. For each *bootinfo object*, present a choice corresponding to each valid `<chrp-boot>` section contained therein. For each such choice, allow the user to either:
  - Execute the contents of that *bootinfo object's* `<boot-script>` element.
  - Set the **"boot-command"** configuration variable to the contents of that *bootinfo object's* `<boot-script>` element.
- Present a choice corresponding to each install device, which, when invoked, will attempt to locate a bootinfo object at the device's standard location (see Table 1).
- Allow the user to manage configuration variables
- Allow the user to invoke the Open Firmware user interface

Additional options that could be implemented would be to provide a means to get to diagnostics or specific platform options.

There *shall* be at least one key sequence to enter the multi-boot platform function for a CHRP platform.

> **Note: Operating System have the responsibility to update the NVRAM System Partition Variable to reflect a change where the *bootinfo.txt* file is located; e.g., moving to a different disk device. Also, the OS is responsible for maintaining the contents of the *bootinfo.txt* file.**

3.2. Reboot-Command Variable Description

The OS can cause Open Firmware to execute a specified sequence of commands at the next boot by setting the value of the **"reboot-command"** configuration variable. CHRP firmware implementations *shall* implement the following configuration variable.

reboot-command(-- addr len) **N**

One time or temporary reboot command.

The value of this configuration variable is a string consisting of zero or more lines of text, with lines separated by either <return>, <linefeed>, or <return><linefeed>.

During firmware start-up, just prior to checking the **auto-boot?** configuration variable for automatic booting, the firmware *shall* check the value of **reboot-command**. If the value is not the empty string, the firmware *shall* save the value to a temporary location, set **reboot-command** to the empty string, and evaluate the saved value as though it were a series of user interface command lines.

If the evaluation of **reboot-command** returns without executing, the firmware *shall* proceed with its normal start-up sequence. In typical usage, however, the value of **reboot-command** will include a **boot** command that starts a client program and does not return.

4. CHRP PowerPC Processor

Open Firmware defines a minimum cell size of 32 bits; therefore, only one cell is necessary to represent addresses up to 4GB (32 bits). Two cells are necessary to represent addresses above 4GB and within 64 bits.

4.1. Processor Endian-ness Support

An implementation of the PowerPC Open Firmware *shall* support both Big- and Little-Endian system implementations. This section describes added features to the core Open Firmware architecture features to support bi-endian booting.

4.1.1. Bi-Endian Booting

The Configuration Variable **`little-endian?`** must be implemented. The basic concept of bi-endian support is to keep in the **`little-endian?`** variable a "cached" indication of the desired endian-ness of client programs (for example operating systems or their loaders). This variable indicates the expected endian-mode of a boot target; false (0) indicates Big-Endian, true (-1) indicates Little-Endian; the default value of **`little-endian?`** is implementation dependent.

The client program must describe its endian-mode in the header section of its image as described in Section 10.4. on page 50. When Open Firmware is started, Open Firmware *shall* use the value of **`little-endian?`** to establish the endian-mode of a system. After Open Firmware locates and loads a client program, the correct endian-mode must be verified with the description in the header section of the client program image. If the cached value is correct, Open Firmware proceeds, with the system in its current endian-mode.

If, however, Open Firmware determines that the endian-mode of the client program is different from what it had assumed, it must set **`little-endian?`** appropriately and reconfigure the system (hardware and firmware) for the new endian-mode, possibly by resetting the system as with reset-all.

> **Note: The endian-mode applies to the entire hardware platform, including the processor(s). Open Firmware *shall* perform whatever steps are required to enable the platform to run in the specified mode.**

Client programs can use `setprop` to alter the value of **`little-endian?`**; users can alter it via the `setenv` command from the user interface (if present). The alteration of **`little-endian?`** *shall* not cause the platform to be reconfigured until the platform is re-booted.

> **Note: This mechanism introduces an extra configuration pass. However, this occurs only when switching the endian-mode from that which was last used. For most boots, Open Firmware will be appropriately configured, so that no additional overhead will occur.**

5. Open Firmware Platform Extensions

This section defines Open Firmware properties, methods, device tree structure and Client Interface Service requirements for a CHRP platform.

5.1. Open Firmware Root Node

This section defines additional properties and methods associated with CHRP platforms that CHRP operating systems expect to find in the root node. The unit address of all non-system nodes that are children of the root node *shall* have the same value each time the platform is booted; i.e., *shall* be invariant for each boot process.

> **Note: This requirement ensures that the PHB would have a stable unit address. Violations of this rule may require reinstallation of an OS.**

> **Note: Typically, the root node corresponds to the common processor bus in a PowerPC platform.**

5.1.1. Root Node Properties

This section defines the additional properties which *shall* be present in the root node unless otherwise specified.

`"#address-cells"` **S**

Standard *property*, encoded as with **encode-int**, that specifies the number of cells required to represent physical addresses on the processor bus. The value of **"#address-cells"** for the processor bus *shall* be 1 or 2 depending on whether there is any area addressable above 4GB's.

"clock-frequency" **S**

Standard *property*, encoded as with **encode-int**, that represents the primary system bus speed (in hertz).

"system-id" **S**

Standard *property*, encoded as with **encode-string**, that contains the identification of the computer system(Reference the **"name"** property in the core specification[1]). This string should be unique across all systems and all manufacturers. An example of an address of this form is "0nnnnnnmmmmmm" where nnnnnn is a sequence of 6 uppercase hexadecimal digits representing a 24-bit value that identifies manufacturer and mmmmmm is a sequence of 6 uppercase hexadecimal digits representing a 24-bit binary number assigned by the manufacturer to assure uniqueness.

> **Note: For platforms with built-in ethernet or other IEEE 802-style interfaces, the 6-byte MAC address assigned to that interface meets the requirements and could be used as the system-id.**

"model" **S**

prop-name is a printable string identifying the manufacturer and model number of the platform.

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

The value of this property is a vendor dependent string which identifies this platform via its manufacturer and model number.

"device_type" **S**

prop-name is a printable string identifying the platform as CHRP Compliant.

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

The value of this property is a string, "chrp" which identifies the platform is CHRP Compliant.

"pc-emulation" **S**

> **Note: The recommended procedure to determine if the PC Emulation Function is present is to do a "test-method" (Refer to Section 10.5.2. on page 54) for the set-pc-emulation method. This property could be deleted in future CHRP Open Firmware Architecture Versions.**

prop-name indicates the presence of the PC Emulation option for PCI Host Bridge 0 and the Memory Controller(s).

prop-encoded-array: <none>

This property *shall* be present if PCI Host Bridge 0 and the Memory Controller(s) support the PC emulation option, otherwise this property *shall* be absent. (The option may be enabled via the **set-pc-emulation** method, if the PC Emulation option is present, see Section 5.1.2. on page 25.)

"exception-relocation-size" **S**

prop-name specifies the fixed amount of address space above physical address 0xFFF00000 which is relocated down into System Memory when the PC Emulation option is enabled.

prop-encoded-int: Integer, encoded as with **encode-int**.

The granularity of **"exception-relocation-size"** properties value is 4 KB and the minimum size is 12 KB. This property *shall* be present if the PC Emulation capability is available, otherwise this property *shall* be absent.

"processor-hole" **S**

> **Note: The recommended procedure to determine if the Processor Hole Function is present is to do a "test-method" (Refer to Section 10.5.2. on page 54) for the "set-processor-hole" Method. This property could be deleted in future CHRP Open Firmware Architecture Versions.**

prop-name indicates that this PCI Host and Memory Controller has the optional processor-hole capability.

prop-encoded-array: <none>

This property *shall* be present if PCI Host Bridge 0 and the Memory Controller(s) support the processor-hole, otherwise this property *shall* be absent. (The processor hole may be enabled via the **`set-processor-hole`** method if the capability is supported, see Section 5.1.2. on page 25.)

`"platform-open-pic"` **S**

prop-name indicates the system Open Firmware Interrupt Controller physical addresses.

prop-encoded-array: List of system *phys-addr* values

This property value is a list of system physical addresses corresponding to **"reg"** property of the system Open Firmware Interrupt Controller

5.1.2. Root Node Methods

This section defines methods associated with the platform via "/" (the *root* node).

`set-processor-hole ( ``true | false`**` -- )`** **M**

Enable(true) or disable(false) the processor-hole on PCI Host Bridge 0 and the Memory Controller(s). This method *shall* be present if the processor-hole capability is supported.

`set-pc-emulation ( ``err-value tem-value enable?`**` -- )`** **M**

If the flag *enable?* is true, enable the PC Emulation option on PCI Host Bridge 0 and the Memory Controller(s). Also disable the system-memory-alias on PCI Host Bridge 0 and the Memory Controller(s). The input value *err-value* is loaded into the Exception Relocation Register (ERR) in the PCI Host Bridge(s) and Memory Controller(s). The input value *tem-value* is loaded into the Top of Emulated Memory (TEM) boundary in the PCI Host Bridge(s) and Memory Controller(s). If the flag *enable?* is false, the PC Emulation option on PCI Host Bridge 0 and the Memory Controller(s) is disabled. The state of PCI Host Bridge 0's initial memory aliases is undefined after it is disabled. This method *shall* be present if the PC Emulation option is present. The units for *err-value* and *tem-value* are in bytes.

5.1.3. ROM Node(s)

The ROM Node(s) *shall* be a child or children of the root node.

5.1.3.1. ROM Node Properties

Each ROM Node *shall* have the following properties:

`"name"` **S**

Standard *prop-name*: The value of this property *shall* be `"rom"`.

`"reg"` **S**

Standard *prop-name* to define a *unit-address* for the node.

prop-encoded-array: One (*phys-addr, size*) pair.

The *phys-addr* of this property *shall* be the starting physical address of this ROM and the *size* value *shall* be 0. The *size*=0 prevents a conflict with the **`"reg"`** of this node's children.

`"#address-cells"` **S**

Standard *prop-name* to define the address space representation of child nodes.

prop-encoded-array: an integer, encoded as with **`encode-int`**. Its value *shall* be identical to that of this node's parent's **`"#address-cells"`** value.

`"ranges"` **S**

Standard *prop-name* to define the address range that is decoded by this **`/rom`** node.

prop-encoded-array: One (*child-phys*, *size*, *parent-phys*) triple, where *child-phys* equals *parent-phys* and the number of cells of each corresponds to the parent's `"#address-cells"` value.

"available" **S**

Standard *prop-name*, to define available ROM resources.

prop-encoded-array: Arbitrary number of *phys-addr, size* pairs. *Phys-addr* is a *phys.hi...phys.lo* list of integers, each integer encoded as with **encode-int**. *Size* is one or more integers, each encoded as with **encode-int**.

The value of this property defines resources, managed by this package, that are currently available for use by a client program.

"write-characteristic" **S**

Standard *prop-name*, defines the ROM Technology.

prop-encoded-array: a string, encoded as with **encode-string,** where the value could equal `"flash"`, `"eeprom"`, `"rom"` or `"nvram"`.

"cacheable" **S**

Open Firmware standard property indicating that the ROM is cacheable.

prop-encoded-array: <none>.

The presence of this property indicates that the ROM is cacheable.

5.1.3.2. ROM Node Methods

If one or more ROM nodes are present, they *shall* each implement the following standard methods. The **"reg"** property is used to determine which ROM the standard methods apply to for multiple ROM's.

The following methods must be defined by **/rom** node.

open (-- true) **M**

Standard method to prepare the ROM Node for subsequent use.

close (--) **M**

Standard method to close the previously opened ROM Node.

probe (--) **M**

Standard Open Firmware method used at boot time to probe all ROM's.

The **probe** method for ROM Nodes *shall* probe for FCode images within the address space defined by its **"reg"** property as defined herein. For each page within its address space, look for a valid FCode image. A valid FCode image is defined to start with an FCode-header(See section 5.2.2.5 in [1]) where the first byte is **start1**, the format byte is 0x08, the length field indicates that the FCode program is contained within the address space of the **/rom** node, and where the checksum is correct. (This probing must take into account the possibility that the ROM image is in the opposite endian-ness from which Open Firmware is currently running.)

If such an FCode image is found, a new child node *shall* be created by executing **new-device** and **set-args**, the FCode image copied to memory (taking into account the endian-ness) and the copy evaluated with **byte-load**. (The FCode program can use **my-unit** to create its **"reg"** property.)

5.1.4. ROM Child Node(s)

This section describes the properties and methods for a ROM Child Node.

5.1.4.1. ROM Child Node Properties

The following properties must be created by **/rom** child nodes.

"name" **S**

Standard *prop-name* that names the child node.

Some physical ROM implementations may not fully decode their entire address range. This could lead to multiple images of the ROM to appear at different addresses, due to the "aliasing" of the ROM image. To prevent multiple device nodes from appearing in the device tree, the FCode for such ROMs should look for an already existing peer node that represents their image. This could be done, for example, by checking that any of the **peer** of the **child** of its **parent** node has a **"name"** property value that is the same as this node's FCode would create.

If such a node is found, the FCode should "abort" the evaluation of its FCode (e.g., by executing an **end0**) before creating its **"name"** property. Open Firmware *shall* remove a node when the FCode evaluation for the node does not result in a **"name"** property being defined.

"reg" **S**

Standard *prop-name* that defines the address range of the child node.

5.1.4.2. ROM Child Node Methods

The following methods must be defined by **/rom** child nodes.

open **(** `-- true` **)** **M**

Standard method to prepare this device for subsequent use.

The **open** method must be prepared to parse **my-args** for the case(s) when the node is being opened in order to access "files"; e.g., when the `bootinfo.txt` file is being accessed during the *multiboot menu*.

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

Standard method to close the previously opened device.

load **(** `addr -- len` **)** **M**

Standard method to load an image. The image must be one that is recognized by the Open Firmware **init-program** method. It is strongly recommended that the ELF format be used, since it has the mechanism to specify configuration variable requirements of an OS.

5.2. Run Time Abstraction Services (RTAS) Node

This system node is a child of **"/"** (root). This section defines properties and methods for the RTAS node. The RTAS Node *shall* not have **"reg"** or **"ranges"** properties.

5.2.1. RTAS Node Properties

This section describes the *rtas* node properties.

"name" **S**

Standard *prop-name:* Denotes the RTAS node.

prop-encoded-array: A string, encoded as with **encode-string**

The value of this property *shall* be "`rtas`".

"rtas-event-scan-rate" **S**

prop-name, is the rate at which an operating system should read indicator/sensor/error data

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

The value of this property *shall* be a number indicating the desired rate for reading sensors and/or error information in calls per minute. This number is platform dependent.

"rtas-indicators" **S**

prop-name, indicates indicators implemented.

prop-encoded-array: An array of paired integers(*token maxindex*), each encoded as with **encode-int**.

The values for this property is a list of integers that are the token values(*token*)for the defined indicators and the number of indicators(*maxindex*) for that token which are implemented (Refer to [3], Chapter 7) on the platform.

Note: The indicator indices for a given token are numbered 0 ... maxindex-1.

"rtas-sensors" **S**

prop-name, indicates sensors implemented.

prop-encoded-array: An array of paired integers(*token maxindex*), each encoded as with **encode-int.**

The values for this property is a list of integers that are the token values(*token*)for the defined sensors and the number of sensors(*maxindex*) for that token which are implemented (Refer to [3], Chapter 7) on the platform.

Note: The sensor indices for a given token are numbered 0 ... maxindex-1.

"rtas-version" **S**

prop-name, describes version information for the RTAS implementation.

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

The value of this property *shall* denote the version the RTAS implementation. For this version, the integer *shall* be as defined in [3].

"rtas-size" **S**

prop-name, is the size of the RTAS memory image.

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

The value of this property *shall* be the amount of contiguous real system memory required by RTAS, in bytes.

"rtas-display-device" **S**

prop-name, identifies RTAS Display Device

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

The value of this property *shall* be the *phandle* of the device node used by the RTAS display-character function.

"rtas-error-log-max" **S**

prop-name, identifies maximum size of an extended error log entry.

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

The value of this property *shall* be the maximum size of an extended error log entry, in bytes.

5.2.2. RTAS Function Property Names

This section defines the property names associated with the various RTAS functions defined in the RTAS chapter of the CHRP specification [3].

"restart-rtas" **S**

"nvram-fetch" **S**

"nvram-store" **S**

"get-time-of-day" **S**

"set-time-of-day" **S**

"set-time-for-power-on" **S**

"event-scan" **S**

"check-exception" **S**

"read-pci-config"	**S**
"write-pci-config"	**S**
"display-character"	**S**
"set-indicator"	**S**
"get-sensor-state"	**S**
"set-power-level"	**S**
"get-power-level"	**S**
"assume-power-management"	**S**
"relinquish-power-management"	**S**
"power-off"	**S**
"hibernate"	**S**
"suspend"	**S**
"system-reboot"	**S**
"cache-control"	**S**
"freeze-time-base"	**S**
"thaw-time-base"	**S**
"stop-self"	**S**
"start-cpu"	**S**

For each of the indicated *prop-nam*es, the value returned for the property is a *token.*

prop-encoded-array: The value, *token*, is an integer encoded as with **encode-int.**

If an RTAS function is implemented, there is a property name which corresponds to its function name. The value of this property is a *token*. This *token,* when passed to RTAS via its *rtas-call* interface (see below), invokes the named RTAS function. If a RTAS function is not implemented, there will not be a property corresponding to that function name. See the RTAS chapter of the CHRP Specification [3] for more information about RTAS functions.

5.2.3. RTAS Node Methods

The **instantiate-rtas** method is invoked by the operating system to instantiate the RTAS functionality. This is accomplished via the call-method *Client Interface Service*.

instantiate-rtas (rtas-base-address -- rtas-call) **M**

Invoking the **instantiate-rtas** method binds the RTAS environment to a given location in System Memory and initializes the RTAS environment. The in parameter, *rtas-base-address*, is the physical address to which the RTAS environment is to be bound. This call indicates that RTAS is instantiated in a 32-bit mode. The amount of contiguous real memory that should be allocated for the RTAS environment is given by the value of the **"rtas-size"** property.

Upon completion of the **instantiate-rtas** method, an entry point address, *rtas-call*, is returned. The value of *rtas-call* specifies the physical address of the entry point into RTAS for future RTAS function calls.

instantiate-rtas-64 (rtas-base-address -- rtas-call) **M**

Invoking the optional **instantiate-rtas-64** method binds the RTAS environment to a given location in System Memory and initializes the RTAS environment. The in parameter, *rtas-base-address*, is the physical address to which the RTAS environment is to be bound. This call indicates that RTAS is instantiated in a 64-bit mode. The amount of contiguous real memory that should be allocated for the RTAS environment is given by the value of the **"rtas-size"** property.

Upon completion of the **instantiate-rtas-64** method, an entry point address, *rtas-call*, is returned. The value of *rtas-call* specifies the physical address of the entry point into RTAS for future RTAS function calls.

5.3. PCI Host Bridge Nodes

This section describes the PCI Host Bridge properties which are added or modified for a PowerPC Microprocessor CHRP implementation. Refer to the *PCI Bus binding* [6] for the base PCI properties and methods. If the platform has multiple PCI Host Bridges, then a **"reg"** property *shall* be present in each PCI Node.

> **Note: Since the RTAS PCI configuration access services do not have separate arguments identifying the PCI host bridge to which a service applies, platforms with multiple PCI host bridges must assign them unique bus numbers. An operating system must not reassign those bus numbers if it expects to make subsequent use of the RTAS PCI configuration access services..**

5.3.1. PCI Host Bridge Properties

For each PCI Host Bridge (PHB) in the platform (called a PCI Bus Controller in the PCI Bus binding), a PCI Host Bridge Node *shall* be defined as a child node of the system bus, in accordance with the *PCI Bus binding*[6] and a **"reg"** property that is unique and persistent for each boot. The following properties are modified or added by the PowerPC Microprocessor CHRP architecture and *shall* apply to each of these nodes.

Each PHB *shall* also have the **"used-by-rtas"** property, since rtas is used for PCI Configuration.

"ranges" **S**

Standard *prop-name*, defines this PHB's physical address ranges.

prop-encoded-array: From two to four (*child-phys, parent-phys, size*) specifications.

This property is mandatory for PCI Host Bridges in PowerPC Microprocessor CHRP implementations. The property value consists of four (*child-phys, parent-phys, size*) specifications, as described in the core document[1].

The first specification *shall* specify the configured address and size of this PHB's I/O Space. (I/O Space is shown as "BIOn" to "TIOn" in the Common Hardware Reference Platform Architecture [3].) The second specification *shall* specify the configured address and size of this PHB's Memory Space. (Memory Space is shown as "BPMn" to "TPMn" except for PHB0, when aliases are enabled, maps BPM0 to BIM, in the Common Hardware Reference Platform Architecture.) If present, the PHB method **change-address-map** can be used to reallocate these areas.

The third and fourth(components) specifications only apply to PHB 0. The third specification *shall* specify the configured address and size of the initial memory aliases. (The initial memory aliases are shown as "BIM" to "TPM0" in the PowerPC Microprocessor CHRP Architecture.) The third specification *shall* have a size of zero if the aliases are not enabled and *shall* be omitted if the aliases and the processor-hole are not enabled. The PHB method **set-initial-aliases** can be used to enable or disable the initial aliases. The fourth specification *shall* specify the configured address and size of the processor-hole. The fourth specification *shall* have a size of zero if the processor-hole is not enabled and *shall* be omitted if the processor-hole is not present. The root node method **set-processor-hole** can be used to enable or disable the processor-hole, if present.

"model" **S**

Standard *prop-name*, indicating this PHB's manufacturer, part number, and revision level. This property *shall* be present if this PHB does not supply the following standard PCI configuration properties which represent the values of standard PCI configuration registers: **"vendor-id"**, **"device-id"**, and **"revision-id"**.

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

The value of this property is a vendor dependent string which uniquely identifies this PHB and is correlated to its manufacturer, part number, and revision level. (see *Core* document[1] for more information.) The string value is device dependent, but *shall* supply information sufficient to identify the part to a level equivalent to the level achievable via the standard PCI configuration registers: **"vendor-id"**, **"device-id"**, and **"revision-id"**.

"initial-memory-aliases" **S**

prop-name indicates the status of the 16 MB initial memory aliases.

prop-encoded-array: <none>

This property *shall* be present if the initial memory aliases are enabled, otherwise this property *shall* be absent. (The aliases can be enabled or disabled via the **set-initial-aliases** method, see Section 5.3.2. on page 31.)

"64-bit-addressing" **S**

prop-name indicates this PHB's capability to address more than 4 Gigabytes of memory.

prop-encoded-array: <none>

This property *shall* be present if this PHB supports addressing more than 4 Gigabytes of memory, otherwise the property *shall* be absent. (If supported, the capability can be enabled via the **set-64-bit-addressing** method, see Section 5.3.2. on page 31.)

"external-control" **S**

prop-name indicates this PHB's ability to support the PowerPC external control facility.

prop-encoded-int: List of integers, each encoded as with **encode-int**.

The property value, if present, is a list of Resource ID's the version of the PowerPC external control facility supports. This property *shall* be present if this PHB supports the PowerPC external control facility, otherwise the property *shall* be absent.

"64-bit-dma" **S**

prop-name indicates this PHB's ability to support Dual Address Cycle for DMA.

prop-encoded-array: <none>

This property *shall* be present if this PHB supports Dual Address Cycle for DMA, otherwise the property *shall* be absent.

"io-hole" **S**

prop-name indicates that this PHB has enabled the io-hole capability, and the size of the hole.

prop-encoded-int: Integer, encoded as with **encode-int**.

This property *shall* be present if this PHB (phb,0 only) supports the io-hole, otherwise the property *shall* be absent. (If supported, the hole can be enabled via the **set-io-hole** method, see Section 5.3.2. on page 31.). The value of this property *shall* be the size of the io-hole in bytes. If the value is 0, the io-hole is disabled.

"reserved" **S**

Standard *prop-name*, defines the physical addresses and sizes of the reserved areas within this PHB's addressable ranges.

prop-encoded-array: Array of *(phys-addr, size)* specifications.

The property values, if present, are the physical addresses and sizes of any reserved areas within this PHB's addressable ranges, as seen by a processor. This property *shall* not exist if there are no reserved areas within this PHB's addressable ranges. Reserved areas should only be used for devices which consume address space in this PHB's addressable ranges but are not otherwise reported in the device tree.

"8259-interrupt-acknowledge" **S**

prop-name indicates the address of the 8259 interrupt acknowledge facility.

prop-encoded-array: One *phys-addr* value

Phys-addr is encoded as with **encode-phys**.

The presence of this property indicates the presence of an 8259 interrupt acknowledge facility. A processor *load-byte* to this address will generate a Interrupt Acknowledge on the 8259 interrupt acknowledge facility. There *shall* be a corresponding **"reserved"** property to remove this address from the bus to which it is attached.

5.3.2. Methods for PCI Host Bridge Nodes

The following *methods* are provided for the purposes of enabling, initializing, and/or modifying PHB features. If these features are present, the node *shall* implement the following methods. Operating systems can call these methods via the *client interface service* call-method function.

change-address-map **(** new-mem-addr size new-io-addr size -- okay? **)** **M**

Relocate a PHB's peripheral spaces in the system address map, if possible.

Note: The "ranges" property may no longer reflect address spaces after execution of change-address-map and Open Firmware may no longer be capable of performing I/O.

set-discontiguous-io **(** true | false **--** **)** **M**

Enable (true) or disable (false) a PHB's discontiguous I/O map.

Note: Once discontiguous I/O mode is enabled, the client should not call Open Firmware for I/O functions, since Open Firmware is not required to support the discontiguous I/O map.

set-64-bit-addressing **(** location size enable? **--** **)** **M**

If the flag *enable?* is true, enable this PHB's 64-bit addressing capabilities and change the size and location of the PHB's TCE Table. If the flag *enable?* is false, disable this PHB's 64-bit addressing capabilities. For the case where all devices are 64-bit capable, both location and size can be specified as zero, in which case the 64-bit addressing capabilities will be enabled, but no TCE Table will be allocated. The Client Program *shall* initialize and maintain the Open Firmware and RTAS Areas to be mapped 1:1 as long as these areas are in use.

Note: The set-64-bit-addressing method should not be executed if the PHB Device is not open.

set-initial-aliases **(** true | false **--** **)** **M**

Enable (true) or disable (false) the initial memory aliases for this PHB (PHB,0 only).

set-io-hole **(** true | false **--** **)** **M**

Enable (true) or disable (false) the io-hole for PHB 0. This method *shall* be present if the io-hole capability is supported.

5.4. Memory Controller Nodes

This section describes *memory-controller* nodes and their properties.

5.4.1. Memory Controller Node Properties

For each Memory Controller in the platform, a *memory-controller* node *shall* be defined as a child of **"/"** (the root) and *shall* not have a **"ranges"** property. The following properties *shall* apply to each of these nodes.

A Memory Controller can also have the **used-by-rtas** property (See "Miscellaneous Node Properties" on page 34.), if it has functions abstracted by RTAS.

"device_type" **S**

Open Firmware standard property.

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

The value of this property *shall* be "memory-controller".

"reg" **S**

Standard *prop-name*, defines the base physical address and size of this Memory Controller's addressable register space.

prop-encoded-array: One (*phys-address, size*) pair where *phys-address* is encoded as with **encode-phys**, and *size* is encoded as with **encode-int**.

The property value *shall* be the base physical address and size of this Memory Controller's register space.

"model" S

Standard *prop-name*, indicating this Memory Controller's manufacturer, part number and revision level.

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

The value of this property is a vendor dependent string which uniquely identifies this Memory Controller and *shall* be correlated to its manufacturer, part number, and revision level. (see Core document for more information.)

"external-control" S

prop-name indicates this Memory Controller's ability to support the PowerPC external control facility.

prop-encoded-int: List of integers, each encoded as with **encode-int**.

The property value, if present, is a list of Resource ID's the version of the PowerPC external control facility supports. This property *shall* be present if this Memory Controller supports the PowerPC external control facility, otherwise the property *shall* be absent.

"error-checking" S

Standard *prop-name*, defines the error checking capability of the node.

prop-encoded-array: a string, encoded as with **encode-string,** where the value could equal "none", "ecc", or "parity".

5.5. Interrupt Controller Nodes

This section describes the properties for the PowerPC Microprocessor CHRP interrupt controller node. An interrupt controller node *shall* not have the **"used-by-rtas"** property.

5.5.1. Open PIC Interrupt Controller Node Properties

An *open-pic* node *shall* be defined as a child of the bus to which the Open PIC is attached and through which registers are addressed. The following properties apply to this node.

"name" S

Standard *prop-name:* The value of this property *shall* be "open-pic".

"device_type" S

Open Firmware standard property.

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

The value of this property *shall* be "open-pic".

"reg" S

Standard *prop-name*, defines the base physical address(s) and size(s) of this Open PIC's addressable register space.

prop-encoded-array: List of *(phys-addr, size)* specifications.

Phys-addr is encoded as with **encode-phys**, and *size* is encoded as with **encode-int**.

The first entry in this list *shall* be the physical address and size decoded by the base Open PIC Interrupt Delivery Unit (IDU). Successive entries in this list *shall* be the physical addresses and sizes decoded by any additional Open PIC Interrupt Source Units (ISU).

"compatible" S

Standard *prop-name*, to define alternate **"name"** property values.

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

The property value *shall* include "chrp,open-pic".

"interrupt-ranges" **S**

Standard *prop-name*, defines the interrupt number(s) and range(s) handled by the base Open PIC IDU and each additional Open PIC ISU, if any.

prop-encoded-array: List of *(int-number, range)* specifications.

Int-number is encoded as with **encode-int**.

Range is encoded as with **encode-int**.

The first entry in this list *shall* be zero (0) and the number of interrupts handled by the base Open PIC IDU. Successive entries in this list *shall* be the lowest interrupt number and the number of interrupts handled by each additional Open PIC ISU, if any. Successive entries *shall* include an equal or greater interrupt number. It is assumed that if *range* is zero in any entry, that unit does not handle any interrupts in that range. Note that there is a one-to-one correspondence between **"reg"** entries and **"interrupt-ranges"** entries.

"model" **S**

Standard *prop-name*, indicating this Open PIC's manufacturer, part number, and revision level.

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

The value of this property *shall* be a string which uniquely identifies the interrupt controller and *shall* be correlated to the manufacturer, part number, and revision level. This value is device dependent (See the Core document for more information.).

5.6. Additional Node Properties

Additional properties and methods are defined in this section for PowerPC Microprocessor CHRP binding adapters and/or devices.

5.6.1. Interrupt Properties

The properties in this section *shall* be implemented for any device that can present an interrupt for a CHRP platform implementation.

5.6.1.1. Interrupt Tree Node Properties

The CHRP Platform *shall* adhere to the *Open Firmware: Recommended Practice - Interrupt Mapping(*Refer to [14] for an interrupt structure Open Firmware representation) definition of the interrupt structure. There are four properties defined by the recommended practice for the interrupt structure.

"interrupt-controller" **S**

Standard *property-name* to define the Open Firmware Interrupt Controller..

"interrupts-map" **S**

Standard *property-name* to define the interrupt mappings.

"#interrupt-cells" **S**

Standard *property-name* to define the number of cells in its child interrupt space.

5.6.1.2. Interrupt Child Properties

The following properties are defined for children of an interrupt nexus node.

"interrupt-parent" **S**

Standard *property-name* to denote the interrupt tree parent of this note.

5.6.2. Miscellaneous Node Properties

This section defines properties which support bridge, adapter and device characteristics. These properties *shall* be present for a CHRP platform.

"slot-names" **S**

prop-name: Describes external labeling of adapter/device connectors.

prop-encoded-array: An integer, encoded as with **encode-int,** followed by a list of strings, each encoded as with **encode-string.**

The integer portion of the property value is a bitmask of available connectors; for each connector associated with the adapter/device, the bit corresponding to that connector's ID number is set from least-significant to most-significant ID number. The number of following strings is the same as the number of connectors; the first string gives the platform nomenclature or label for the connector with the smallest ID number, and so on.

Note: Each device that has a connector should identify the order and contents of the list of strings in a binding.

"built-in" **S**

Standard *prop-name:* Any supplied device/part by a CHRP platform that plugs into an industry standard I/O expansion bus; i.e., PCI and ISA.

prop-encoded-string: <none>.

Note: This property will also include platform 'riser' cards.

"used-by-rtas" **S**

Standard *prop-name:* Indicates the device can be in use by an RTAS Function Call.

prop-encoded-int: Presence of property indicates that I/O Device can have the ability to have an I/O or resource conflict with a RTAS Function Call.

Note: An OS typically will not load device driver if "used-by-rtas" property is present.

5.7. Aliases Node

A *device alias*, or simply *alias*, is a shorthand representation of a *device-path*. *Aliases* are properties of the *aliases* node, encoded as with **encode-string**. Aliases are typically used by a user to facilitate not specifying a long path name at the User Interface 'ok' prompt.

An implementation of Open Firmware for a CHRP platform *shall* provide the following aliases as properties of the *aliases* node, if the corresponding device exists:

"disk" **S**

prop-name, indicating the device path of the factory default disk.

"tape" **S**

prop-name, indicating the device path of the factory default tape.

"cdrom" **S**

prop-name, indicating the device path of the factory default CDROM.

"keyboard" **S**

prop-name, indicating the device path to the keyboard to be used for the User Interface.

"mouse" **S**

prop-name, indicating the device path to the mouse to be used for the User Interface.

"screen" **S**

prop-name, indicating the device path to the screen to be used for the User Interface.

"pc-keyboard" S

prop-name, indicating the device path of the factory default PC-style keyboard.

"pc-mouse" S

prop-name, indicating the device path of the factory default PC-style mouse.

"adb-keyboard" S

prop-name, indicating the device path of the factory default ADB-style keyboard.

"adb-mouse" S

prop-name, indicating the device path of the factory default ADB-style mouse.

"scsi" S

prop-name, indicating the device path of the factory default built-in SCSI device.

"com1" S

prop-name, indicating the device path of the factory default 16550-style serial port known as "com1."

"com2" S

prop-name, indicating the device path of the factory default 16550-style serial port known as "com2."

"scca" S

prop-name, indicating the device path of the factory default SCC-style serial port known as "SCCA."

"sccb" S

prop-name, indicating the device path of the factory default SCC-style serial port known as "SCCB."

"floppy" S

prop-name, indicating the device path of the factory default floppy drive.

"net" S

prop-name, indicating the device path of the factory default built-in network interface controller.

"rtc" S

prop-name, indicating the device path of the factory default real-time-clock chip.

"nvram" S

prop-name, indicating the device path of the factory default NVRAM.

5.8. Event Sources Node

The Event-Sources Node describes the possible RTAS Error and Event Classes for Open PIC Interrupts(Refer to [3], Table 50). The Event-Sources Node *shall* be defined to be a child of the root device tree node. The following properties *shall* be defined for this node:

"name" S

Open Firmware standard property. The value of this property *shall* be `"event-sources"`.

5.8.1. Child nodes of the Event Sources Node

The following specify standard child nodes of the **/event-sources** node. These nodes could be present in a CHRP platform.

5.8.1.1. internal-errors

The presence of the node indicates that all or some of the function has been implemented.

"name" S

Open Firmware standard property. The value of this property *shall* be `"internal-errors"`.

"open-pic-interrupt" **S**

The property signifies that RTAS abstracted event sources will use one or more Open PIC interrupt sources to report the internal-errors class of error conditions.

The value of this property *shall* be a *prop-encoded-array* of integers encoded as with **encode-int**.

5.8.1.2. epow-events

The presence of the node indicates that all or some of the function has been implemented.

"name" **S**

Open Firmware standard property. The value of this property *shall* be `"epow-events"`.

"open-pic-interrupt" **S**

The property signifies that RTAS abstracted event sources will use one or more Open PIC interrupt sources to report events of the epow-events class.

The value of this property *shall* be a *prop-encoded-array* of integers encoded as with **encode-int**.

5.8.1.3. power-management-events

The presence of the node indicates that all or some of the function has been implemented.

"name" **S**

Open Firmware standard property. The value of this property shall be `"power-management-events"`.

"open-pic-interrupt" **S**

The property signifies that RTAS abstracted event sources will use one or more Open PIC interrupt sources to report events of the power-management-events class.

The value of this property *shall* be a *prop-encoded-array* of integers encoded as with **encode-int**.

6. Symmetric Multi-Processors(SMP)

CHRP platforms can have Symmetric Multi-Processor(SMP) Configurations. In addition to the processor node properties defined in [5], *PowerPC Processor binding to: IEEE Std 1275-1994, Standard for Boot (Initialization, Configuration) Firmware*, a SMP Configuration will utilize the /cpus node as explained in section 6.1.

6.1. SMP Platform Device Tree Structure

Open Firmware requires that multiple instances of any device that appears more than once in the device tree must be unique and distinguishable by means of their **"reg"** properties. For CHRP platforms, processors *shall* not be directly attached to the main physical bus(root node (“/”)). Instead, cpu devices *shall* be children of the **/cpus** node.

The /cpus node *shall* have one child node of device type cpu for each processor. The value of the **"reg"** property of each cpu device node *shall* be an integer encoded as with **encode-int**, which *shall* be the index of the output of the Open PIC interrupt controller which is attached to this processor; i.e., the bit number corresponding to this processor in the Open PIC Destination Register. The *ihandle* of the “executing” processor *shall* be published in the **"cpu"** property of the **/chosen** node.

> **Note: The properties of a cpu device are already defined in the PowerPC Processor binding document[5]. The only change for symmetric multiprocessor (SMP) systems is that there will be a cpu device node under the /cpus node for each individual processor. Other properties of the cpu devices *shall* conform with the requirements stated in the PowerPC Processor binding document.**

6.2. SMP Properties

The following properties are for a CHRP PowerPC SMP environment. These SMP properties will be under the **/cpus** Node.

"slot-names" S

prop-name: Describes platform labeling of add-in cpu/processor card slots.

prop-encoded-array: An integer, encoded as with **encode-int,** followed by a list of strings, each encoded as with **encode-string.**

The integer portion of the property value is a bitmask of possible processors; for each add-in slot on the bus, the bit corresponding to that slot's ID number is set from least-significant to most-significant ID number. The number of following strings is the same as the number of slots; the first string gives the platform nomenclature for the slot with the smallest ID number, and so on. The processor's first component of the **"reg"** property can be used as an index into the bitmask integer of this property.

"smp-enabled" S

The presence of this property signifies that the platform is SMP enabled, even if it only has one processor.

prop-encoded-array: <null>

6.3. PowerPC Microprocessor Processor ID(PIR) Register

If a PIR Register is present and the operating system wishes to use it, the operating system is responsible for setting the value. Open Firmware or RTAS *shall* not modify the value.

7. Device Power Management Properties/Methods

This section defines standard platform node properties, device node properties, and methods related to power management. The properties and methods of this section *shall* be implemented on any platform which supports power management except where noted. However, it is still being enhanced. Operating system providers who want to ensure that the data needed for their power management policies is included should contact the authors of this document.

7.1. System Node Properties

The following defines properties are to be associated with the rtas and the power-management-events nodes of the device tree.

7.1.1. Properties assigned to the RTAS node

Power domains are a feature of CHRP platforms which support power management. Within the Open Firmware device tree, power domains are represented by a power domain identifier which is defined to be an integer in the range 0..n-1, where n is the number of power domains on the platform.

"power-domains-tree" S

Standard *prop-name* which defines the power domain hierarchy for this platform.

prop-encoded-array: An array of integers, each encoded as with **encode-int**, that is a flattened representation of the power domain dependency tree.

The array consists of a number of tuples, one for each power domain defined on the platform. Each tuple consists of the power domain identifier domain#, followed by the number of power levels #levels supported by the domain, followed by an array of tuples, one for each level. These tuples consist of a level identifier level, followed by the number of power sources from which the domain draws power, followed by an array of tuples (power-source-id, power). The power domain tuple is terminated by the number of children #children followed by a list of the domain identifiers of each child. The power values are expressed in milliamperes and include only the power consumed by support logic not represented as devices in the device tree including any RTAS abstracted devices within the particular power domain.

"power-domains-controllers" **S**

Standard *prop-name* which defines the power domain controllers (The CHRP architecture calls them power domain control points) present on this platform.

prop-encoded-array: an array of integers, each encoded as with **encode-int**. Each integer is the *phandle* of the device tree node that functions as the power domain controller for a domain. A single controller may serve as the control point for multiple domains. Each device which serves as a controller encodes the controls-power-domain property.

"power-domains-names" **S**

Standard *prop-name* used to define the user readable names for the power domains.

prop-encoded-array: an array of strings, each encoded as with **encode-string**, that are the user readable names for the domains.

The number of strings matches the number of domains and there is a one-to-one correspondence between the entries in the **"power-domain-controllers"** property and the entries in this array.

"platform-power-sources" **S**

Standard *prop-name* defining the platform power sources.

prop-encoded-array: an array of integers, each encoded as with **encode-int.** The array is structured as a number of tuples. Each of these tuples consists of the values source-voltage, (given in millivolts), peak-power, continuous-use-power (both expressed in milliamperes supplied at the stated voltage), and conversion-efficiency (expressed in percent).

"power-sources-names" **S**

Standard *prop-name* defining the platform power source names.

prop-encoded-array: an array of strings, each encoded as with **encode-string**, that are the user readable names for the power sources.

The number of strings match the number of power sources and is in one-to-one correspondence to the entries in the **"platform-power-sources"** property.

"platform-battery-sources" **S**

Standard *prop-name* defining the batteries utilized by a platform.

prop-encoded-array: an array of integers, each encoded as with **encode-int.** Each value in this array is the manufacturer's rated capacity of the battery expressed in milliwatt-hours.

"battery-sources-names" **S**

Standard *prop-name* defining the human-readable identifier of the batteries utilized by a platform.

prop-encoded-array: an array of strings, each encoded as with **encode-string.** Each entry in this array corresponds one-for-one with the batteries defined in the **"platform-battery-sources"** property.

7.1.2. Properties of the power-management-events node

"power-type" **S**

Standard *prop-name* defining the power management event types implemented on a specific platform.

prop-encoded-array: an array of integers, each encoded as with **encode-int.** Each entry in this array is one of the values defined by Reference [3] for the power management event types corresponding to the event types available on the platform.

7.2. Device Properties

"power-domains" **S**

Standard *prop-name*: Indicates the power domains of which this device is a member. If the device is a member of the default power domain 0 alone, this property does not need to be provided.

prop-encoded-array: List of one or more domain-ids to which this device belongs. domain-ids is encoded as with **encode-int**.

The power-domains property should only list the domain-ids of the lowest power domain tree nodes in which this device has membership.

"device-power-states" **S**

Standard *prop-name* which describes the power states this device supports. This property *shall* be provided for each physical device which has multiple power states, if platform firmware provides device power state information.

prop-encoded-array: An array of integers, each encoded as with **encode-int** that defines the supported power states for this device.

The array consists of an integer representing the initial device power state after reset, followed by the number of power sources from which the device draws power, followed by an arbitrary number of tuples, one for each supported power state of the device. Each tuple consists of the state, followed by an array of tuples (power-source-id, power) giving the average power consumption from each power source during active use. This is followed by another array of tuples (power-source-id, power) giving the idle power consumption for each power source. Each power state tuple is terminated by the maximum expected power usage lifetime in seconds for the device if it were to remain in that state. The value power is stated in the millamperes consumed at the voltage supplied by the power source.

The value state *shall* be further constrained to have the following semantics:

Table 2. Semantics of device state values

Value	Semantics
100	This is the device's most responsive state.
20-99	The device is functional. The range represents a range of performance.
11-19	Reserved
10	Device is not operational, but retains its internal functional parameters.
1-9	Reserved
0	Device not functional, may lose internal functional parameters.

The semantics of device power states may be further defined by device type specific bindings.

The interaction of the defined semantics of device power state and domain power level is defined in Table 3. Combinations of Device Power State/Domain Power Level. Those combinations not marked are disallowed.

Table 3. Combinations of Device Power State/Domain Power Level

		Device Power State			
		100	**99-20**	**10**	**0**
Domain Power Level	Full On	Allowed	Allowed	Allowed	Allowed
	Reduced		Allowed	Allowed	Allowed
	Freeze			Allowed	Allowed
	Off				Allowed

"device-state-transitions" **S**

Standard *prop-name* that describes the legal power state transitions supported by the device. This property *shall* be provided if platform firmware provides device power state information.

prop-encoded-array: an array of integers, each encoded as with **encode-int** that defines the legal power state transitions for this device.

The array is structured as a number of tuples, one for each possible transition. Each tuple consists of the starting state, followed by the destination state, followed by an array of tuples (power-source-id, power), one for each power source, followed by the time required to make the transition in microseconds, followed by the maximum count allowed for this transition. The starting state and destination state are values defined in the **"device-power-states"** property. The value power is stated in the millamperes consumed.

"power-sources" **S**

Standard *prop-name* which designates this device as a consumer of power sourced from a defined power source. This property *shall* be provided if platform firmware provides device power state information.

prop-encoded-array: an array of integers, each encoded as with **encode-int** that gives the list of power sources to which this device is connected. The values are indices into the platform-power-sources data structure

"power-management-mapping" **S**

Standard *prop-name*. This optional property provides a device dependent mapping between device power state and commands which the device driver sends to its device. Also provides information concerning which device power states are supported for each of the four domain power levels. See the device type binding for a definition of the property value.

7.2.1. Properties for Power Domain Control Points

The following are specific to devices which can act as power domain control points.

"controls-power-domains" **S**

Standard *prop-name* which designates the domains over which this device exercises control.

prop-encoded-array: an array of integers, each encoded as with **encode-int** that defines the domains for which this device can act a power domain control point.

A single device may serve as multiple logical control points.

7.3. Power Management Related Methods

This section defines methods associated with device tree nodes which serve as power domain controllers (CHRP architecture calls them control points).

set-power-level (domain# level -- actual-level) **M**

This method is only present for power domain controllers. The domain# is the power domain whose power level is altered, and level is the desired level. actual-level reports the level to which the domain was actually set.

get-power-level (domain# -- level) **M**

This method is only present for power domain controllers. The domain# is the power domain that is being queried. level is the current level at which the domain is now operating.

system-off (--) **M**

Method to turn the system off. This method is attached to the root node of the device tree and is only present in a platform with software control over system power.

7.4. Power Management NVRAM Partition

The standard format for data located in the Power Management Configuration partition of the NVRAM is under development(see "Device Power Management Configuration Resources" on page 67).

8. ISA Device Support

The CHRP System binding identifies ISA platform support areas that are unique. These areas have to do with the DMA and Interrupt Controller attributes. In addition, it is desirable to further define certain choices for the CHRP standard ISA I/O and/or ISA pluggable resources where an OS expects to find information defining the attributes. These additional ISA dependencies for a CHRP Platform are in the area of DMA Modes, interrupt request states, address modes(I/O or memory), state of Interrupt and DMA Request Lines (tri-stateable: can be disabled) and the Unit Address Format. Open Firmware is responsible for programming ISA modes that are configured appropriately with the stated level of CHRP Platform support.

The following sections describe further restrictions and/or requirements for the CHRP Platform ISA Devices and/or Adapters:

- ISA Resources
    - ISA address mode
    - Interrupt triggering/level modes
    - DMA modes and width
- ISA Properties
- ISA Unit Address Format
- ISA Configuration Requirements

8.1. Additional ISA Requirements

Additional ISA Adapter/Device resource restrictions and/or formats are stated in this section for a CHRP Platform. Refer to *ISA/EISA/ISA-PnP binding*[7] for the various resources defined in this section. Open Firmware information for ISA is defined in the following Table 4. for a set of standard ISA I/O platform devices or ISA plug-in adapters/devices.

The meaning of the table headings below are defined to be:

- Device Tree - The Open Firmware device tree is expected to define the entries within property values. The operating system is expected to get the properties from the Open Firmware Device Tree.
- Device Binding - The Open Firmware device binding for that device is expected to define the entries within a property value, if necessary. The operating system is expected to get the properties from the Open Firmware device binding for that device.
- CHRP platform - The capabilities of the CHRP legacy interrupt controller or DMA controller create restrictions on the entries within property values, if necessary.

The ISA terms or nomenclature used in the table below relate to the *ISA/EISA/ISA PnP binding*[7] in the following manner:

interrupt level	irq#
interrupt triggering	type
dma channel #	dma#
DMA master or slave mode	busmaster
DMA width	width and countwidth
DMA timing	mode

Table 4. ISA Open Firmware Resource Information

Device Tree	**Device Binding**	**CHRP Platform Restrictions on ISA Adapters/Cards**
Interrupts - ‘level’ specified	Interrupts - ‘triggering’ specified	Interrupts - NO support for ’active high level sensitive’ or ‘high to low edge’ triggering
DMA - ‘channel #’ specified	DMA - master or slave Mode specified DMA - channel and width count	DMA - No support for “C” Mode, 32-bit channel or 32-bit countwidth
Address - physical or aliased address specified	Address Mode - I/O or memory address specified	
DMA: ISA “Compatibility” Mode *shall* be supported(see note 1)	DMA: ISA “Compatibility” Mode *shall* be supported(see note 1)	DMA: ISA “Compatibility” Mode *shall* be supported(see note 1)

Note 1: ISA DMA slaves and masters must support the ISA compatible timing mode. ISA DMA slaves and masters are allowed to support additional timing modes that may be reported in the Open Firmware device tree. An operating system may restrict its mode operation to just the ISA compatible timing mode or may use the timing mode specified in the Open Firmware device tree node for that device.

8.1.1. ISA Addresses

All ISA Adapters *shall* use an I/O Address or memory address for operation with the ISA Bus per Table 4.

8.1.2. ISA Interrupts and DMA Channels

A CHRP Platform *shall* ensure that all I/O device states are consistent with the interrupt legacy controller and DMA controller specified in Table 4.

8.1.3. ISA Unit Address Format

A CHRP platform may use a ‘legacy’ or a ‘PnP’ Unit Address Format defined in Reference[7] as the first entry in a **`"reg"`** property. The ‘legacy’ Unit Address value used by a CHRP platform for I/O ISA devices *shall* remain invariant across platform boots.

This requirement ensures that ISA adapters or functions would have a stable unit address across boots. Violations of this rule may require reinstallation of an OS.

8.1.4. ISA Additional Properties

The CHRP Platform *shall* support two additional properties that identify when an adapter cannot disable interrupt lines or DMA Channels.

"dma-enabled" **S**

prop-name indicates that a DMA Channel is active

prop-encoded-array: a list of integers, each one encoded as with **encode-int**.

The property value is a list of integers for index indicating DMA channel numbers used by the adapter/device. The presence of this property signifies that the specified DMA Channels for the ISA adapter are enabled and thus not tri-stated.

"interrupt-enabled" **S**

prop-name indicates that interrupt request line is active

prop-encoded-array: a list of integers, each one encoded as with **encode-int**.

The property value is a list of integers for index indicating interrupt request lines used by the adapter/device. The presence of this property signifies that the specified interrupt request lines for the ISA adapter are enabled and thus not tri-stated.

9. Configuration of Platform Resources

Any computer platform is composed of standard components which are invariant (platform 'built-in' standard I/O and power management), optional components which are detectable (a second processor, for example), and configurable components which are self-identifying (system memory, for example). Most computer platforms also provide one or more industry standard I/O buses which allow the insertion of specialized functional adapter cards. These buses generally support a method for automatic identification, interrogation, and option selection of installed adapter cards. Some older adapter cards, however, may not support these methods or may supply incomplete information. Thus it is necessary to provide interactive utilities to aid in the setup of 'legacy' pluggable ISA adapter cards and to provide information describing the power management of platform resources.

A CHRP platform *shall* be capable of configuring ISA resources, if required, and power management resources (if implemented). The platform *shall* be capable of using both 'legacy' ISA adapter cards and 'Plug and Play'(PnP) ISA adapter cards (see the *ISA/EISA/ISA PnP binding*[7] and *Device Support Extensions binding*[9]). For the ISA PnP Adapter Cards, Open Firmware is capable of reading data resource records from the adapters to determine properties and device tree nodes. For 'legacy' ISA adapter cards, a configuration support utility is required to interact with the user to provide assignment of ISA resources; i.e., the I/O address, interrupt number, DMA channel, system memory and slot-names. The platform Open Firmware *shall* provide the ISA Configuration Utility(ICU) as defined in section 9.1., if it has ISA slots.

A CHRP Platform *shall* also have the capability of configuring power management resources, if power management is implemented by the platform, as defined in section 9.2.

9.1. ISA Resource Configuration

CHRP platforms *shall* provide an ICU program which the vendor and users of the platform can use to generate the Open Firmware configuration information for ISA devices, as described in this section. The information the ICU produces *shall* be stored in the NVRAM Configuration Partition (NVRAM partition 0x71), and the information *shall* have the format described in this section. In addition, the platform firmware *shall* provide a user interface method (keyboard sequence, icon, or other means) for the user to enter the utility at system reboot time to enter configuration data before an operating system is executed.

9.1.1. ISA Resource Configuration Description

The CHRP Platform ISA Configuration Program *shall* contain appropriate functions which will determine the information needed to completely fill in the required data structures. This may require obtaining information from a human user, as well as interacting with and controlling hardware interface features of some ISA devices. A recommended set of interactions is:

- Interact with CHRP platform user to assign fixed ISA resources, if necessary
    - Determine status of hardware defined ISA adapter resources (switches)
    - Allocate ISA resources per adapter
    - Verify that resources don't conflict with platform base I/O devices
- Interact with CHRP platform user to assign programmable ISA resources, if necessary
    - Set the state of software controlled ISA adapter resources (soft switches)
    - Allocate ISA resources per adapter
    - Verify that resources don't conflict with platform base I/O devices and any fixed ISA adapter resources
- Assign ISA-PnP adapter and/or base device relocatable resources
    - Determine relocatable ISA resources
    - Allocate ISA resources per adapter
    - Verify that the assigned ISA-PnP adapter and/or base device resources do not conflict with ISA (base I/O devices and adapters) assigned resources
- Verify that ISA and ISA-PnP resources do not conflict with the PCI-to-ISA bridge/PCI bus resources
- Interact with CHRP platform user to provide the capability to display platform assigned ISA/ISA-PnP resources

9.1.1.1. ISA Configuration Handle Assignment

Logical device handle numbers will be first assigned, starting with number "1" through "m", to ISA legacy card devices. Resource assignment information gained from user input interactively will be tabulated and conflict checked by the ISA Configuration Utility (ICU). Conflict information will be displayed to the user and it will be the user's obligation to re-adjust legacy card jumpers/switches to resolve conflicts and to provide updated resource information to the ICU. After completion of the assignment of legacy card resources, PnP device resource assignments will be made for logical devices "m+1" through "m+n" via standard PnP isolation and auto-configuration methods. Card slot/connector information will also be solicited interactively by the ICU for both legacy and PnP cards to support POST diagnostic fault location requirements.

9.1.2. ISA Legacy Configuration Data Format Requirements

The ISA Data Format *shall* be in a 'PnP' TAG Format (Refer to [33] and [34] for detailed tag information) and represent allocated resources only. The PnP Vendor extension tag will be used to represent the *slot names/connector* information.

The ISA Configuration Data Format will follow the format conventions which have been defined for ISA Plug and Play (PnP) attach cards. The PnP-like format given below will be used to characterize ISA legacy cards which may also be present in a CHRP platform. The Configuration Manager will solicit additional information via an interactive setup routine to obtain resource information for ISA legacy cards as well as physical attribute information (such as card slot location and connector attributes) for all cards in general.

The following information will be stored in NVRAM to maintain configuration status and to characterize resource requirements for ISA legacy cards. It has been assumed here that handles 1 through m+n have been assigned by the Configuration Manager to logical devices in the attach card set.

Table 5. ISA Card Configuration Directory

# Bytes	Data Field
1	Number of Assigned Legacy Card Handles (1 thru m)
1	Number of Assigned PnP Card Handles (m+1 thru m+n)
6	Reserved
32	Card Slot List (Slot location by handle index)
32	Pointers to Legacy Card Resource Data (handle indexed)
64	Reserved for additional physical attributes

Data stored in NVRAM to characterize ISA legacy cards will make use of the following PnP-like format system of tags for small data items:

Table 6. ISA Card PnP-like Small Data Item Tag Format

Tag	Data Item Type
0x15	Logical Device ID (Assigned Handle)
0x1C	Compatible Resource Requirement
0x23	IRQ Resource Requirement
0x2A	DMA Resource Requirement
0x30	Start Dependent Function(optional)
0x38	End Dependent Function(optional)
0x4B	I/O Resource Requirement(10 bit)
0x4C	I/O Resource Requirement(11 bit)
0x4D	I/O Resource Requirement(16 bit)
0x72	PhysicalAttributes(VendorExtension)-*'slot names'/'connectors'*
0x78	End Tag

Currently, the compatible device ID set for the legacy devices is restricted to that defined for equivalent PnP devices. Unlike PnP cards, however, the compatible device ID for legacy cards cannot be used to identify driver requirements.

Large data items will make use of the PnP-like format tags:

Table 7. ISA Card PnP-like Large Data Item Tag Format

Tag	Data Item Type
0x81	Memory Range Descriptor
0x82	ANSI String ID (optional)
0x86	32-bit Fixed Location Memory Range Description

The minimum base address and maximum base address fields of a tag type 0x81 Memory Range descriptor data item must be defined to be the same value. This is required to assure unambiguous interpretation of the memory range attributes as a resource assignment.

An example of an ISA legacy card resource requirement description structure in NVRAM is as follows:

TAG (0x82)	ANSI String Description (optional)
TAG (0x15)	Logical Device ID (Assigned Handle)
TAG (0x72)	Vendor Extension (Slot Location)
TAG (0x1C)	Compatible Device ID
TAG (0x23)	IRQ Resource Requirement
TAG (0x2A)	DMA Resource Requirement
TAG (0x4B)	I/O Port Resource Requirement (May be more than one)
TAG (0x81)	Memory Range Resource Requirement (May be more than one)
TAG (0x86)	32 Bit Fixed Memory Range Resource Requirement (May be more than one)
TAG (0x78)	End Tag.

The absence of a particular data item type indicates that no resource of that type is required or assigned. The presence of an empty (all zeroes) mask in IRQ/DMA resource data items also indicates that no resource of that type is required.

Because ISA devices can conflict in their use of DMA channels, memory addresses, I/O addresses, and interrupts, a configuration utility may detect conflicts which cannot be resolved by software. In such cases, the ICU needs to interact with a human user to suggest configuration changes such as changing jumper settings. It is recommended that the utility suggest non-conflicting ISA resources which the user could set the device to utilize.

9.2. Power Management Resource Configuration

For a platform which supports device power management, all platform power management related information *shall* be resident in the Open Firmware device tree prior to the transfer phase of software operation (see the definition of transfer phase in [3], Chapter 2). Dummy devices *shall* be placed in the device tree for all standard I/O bus connectors which are not in use to provide a node to assign the slot-names, power-domains, and power-sources properties.

Ultimately, the goal is that pluggable devices would not only identify themselves to platform firmware but would also provide all applicable power management related information. As an interim solution, a utility *shall* be provided either in the platform firmware ROM or supplied as a loadable Open Firmware utility on external media. This utility interacts with a person to obtain power management information concerning plug-in adapters and peripherals.

9.2.1. Power Management Information Utility

Any platform capable of being expanded via the addition of power-managed devices *shall* provide a device power management information utility. The purpose of the utility is to allow a person (end-user or system developer) to enter power management related device properties of plug-in adapters and peripherals which have no mechanism to automatically report this information to firmware or system software. The need for this utility will disappear as standard protocols are developed for interrogating pluggable adapters and devices to provide power management related information.

In the most general case, the devices to be added to a node representing a standard bus or I/O port are in the form of multilevel subtrees. The root of this subtree specifies the path to the node in the device tree where the subtree is to be grafted.

The utility determines the path to the node at which to graft the new devices by interacting with a person to receive the information. The utility uses the **"slot-names"** property to identify the location of the device for which it needs information. For example, the utility might prompt the user with, "Enter the name of the first device attached to the external scsi connector labeled 'SCSI1'."

A data structure describing the subtree is stored in NVRAM. The root node of this subtree contains an **"in-graft-node"** property which specifies the path to the parent node where the subtree is to be grafted into the Open Firmware device tree.

As adapters and devices are enhanced to support the automatic reporting of power management information the parent node would supply a method query-power-management-attributes which can be used by firmware to obtain this information without the need for this utility. Any information obtained by direct device interrogation would override that supplied via the PM NVRAM partition.

9.2.2. PM Configuration Process

When the platform is booted after a configuration change and the newly inserted adapter does not support the automatic reporting of power management information, firmware should prompt the user asking if he wishes to supply this information or potentially forfeit some or all of the power management capabilities of the device.

The utility records the information it obtains in the NVRAM Power Management Configuration Partition (NVRAM Signature of 0x71 and name *pm-config*). On a subsequent reboot, platform firmware uses the information saved in NVRAM to fill out the device tree adding new nodes and their properties, as well as adding properties and updating the values of properties of existing device tree nodes.

9.2.3. PM Configuration Format

The NVRAM power management configuration partition is designed to be accessed primarily by firmware, but the partition is designated global and the format is specified to allow a third party to write a power management information utility which runs on the booted operating system.

The data field of the power management NVRAM partition *shall* be defined as follows:

The data field is composed of a header, followed by a number of fixed length data blocks, and finally a variable length property list area. The length of the header and each data block is 8 bytes. The data blocks use 16-bit integer offsets into the partition as pointers to the data blocks and into the property list area. The base of this offset is the beginning of the partition. This effectively limits the size of the PM configuration area to 64K bytes. If more space is required, additional PM configuration partitions may be provided. Each pointer into the property list area locates the start of a null-terminated string which represents a list of property name/value pairs.

The following table specifies the format of the header:

Table 8. Power Management Configuration Data Header

Field	Size	Description
Version	1 byte	Designates the version of the PM Partition data area format
Subtree_ptr	2 bytes	Pointer to the first data block which describes a device subtree
Property_ptr	2 bytes	Pointer to first data block which describes a property list to be added to the base platform device tree
Reserved	3 bytes	Reserved

The PM Partition data area format value *shall* be 1.

The following table specifies the format of the data blocks:

Table 9. Data Block Format

Field	Size	Description
Block_type	1 byte	Designates the data block type
Data Block Data	7 bytes	Remainder of data block, format specific to data block type

Two data blocks are defined: one defining a device node and a second defining properties to be added to the base platform device tree.

The data block type field *shall* have the value 1 for a data block which describes a device node. The data block type field *shall* have a value 2 for a data block which describes a property.

Table 10. Node Data Block Format

Field	Size	Description
Block_type	1 byte	This field *shall* contain the value 0x01
Prop_list_ptr	2 bytes	Pointer to a null terminated string containing the property list for this node
Child_ptr	2 bytes	Pointer to a data block defining a child node of this node. This pointer will be equal to 0x0000 if this node has no children.
Sibling_ptr	2 bytes	Pointer to a data block defining a sibling node of this node. This pointer will be equal to 0x0000 if this node has no siblings.
Reserved	1 byte	Reserved

Table 11. Property Data Block Format

Field	Size	Description
Block_type	1 byte	This field *shall* contain the value 0x02
Node_path	2 bytes	Pointer to a null terminated string giving the path name of the node to which the designated property list belongs.
Property_list_ptr	2 bytes	Pointer to a null terminated string containing the property list to be assigned to the designated node.
Reserved	3 byte	Reserved

The first node of a subtree *shall* have a **"name"** property equal to "/" and *shall* specify the **"in-graft-node"** property. The child_ptr of this data block points to the first in a list of data blocks which describe the nodes which make up the subtree to be grafted onto the system tree.

The final area of the partition is a set of null terminated strings which represent property name/value pair lists. The last string in this area will be terminated by at least two null bytes. The property list for each node *shall* provide all the required PM properties and their values. These include **"power-domains"**, **"device-power-states"**, **"device-state-transitions"**, **"power-sources"**, **"power-management-mapping"**, and **"controls-power-domains"**.

10. Client Program Requirements

For CHRP platforms, the client program requirements are defined in Section 9 of Reference [5], with the following modifications. Open Firmware Client Programs for a CHRP platform *shall* execute in 32-bit mode with an Open Firmware cell size of 1.

10.1. Load Address

The client's load address is specified by the value of the **load-base** Configuration Variable. The value of **load-base** defines the default load address for *client programs* when using the **load** method. **Load-base** *shall* be a real address in real mode or a virtual address in virtual mode. Note that this address represents the area into which the client program file will be read by **load**; it does not correspond to the addresses at which the program will be executed. All of physical memory from **load-base** to either the start of Open Firmware physical memory or the end of physical memory, whichever comes first, *shall* be available for loading the client program.

> **Note: The load-base address represents the area into which the client program will be read by load and does not correspond to the address at which the program will be executed.**

10.2. Initial Register Values

Table 2, "Initial Register Values", of Reference [5] is modified as follows:

- r3 -- *shall* be 0 on client program entry
- r4 -- *shall* be 0 on client program entry

10.3. I/O Devices State

With the exception of the stdin and stdout devices, Open Firmware *shall* close all devices with the following conditions true:

All Devices - no DMA and not interrupting

Normal I/O Devices - not responding to access PCI and ISA PnP Adapter/Devices

HOST Bridges - responding to config cycles and passing through config cycles to children

RTAS Devices - contract with Open Firmware to leave in state to perform intended function

10.4. Client Program Format

The data format of a client program compliant with this specification *shall* be either ELF (Executable and Linkage Format) as defined by [19], and extended by section 10.4.1.1., or PE (Portable Executable) as defined by [17]. The standard ELF format contains explicit indication as to the program's execution modes (e.g., 32- or 64-bit, Big- or Little-Endian). CHRP only supports the 32-bit version (i.e., ELFCLASS32) for 32 and 64 bit platforms.

Note: other client program formats may be supported, in an implementation specific manner, by an Open Firmware implementation.

A standard client program *shall* be statically linked, requiring no relocation of the image. The program's entry point (e_entry) *shall* contain the address of the first PowerPC instruction of the client program. It is the responsibility of the client program to establish the appropriate value of the TOC (r2), if necessary.

Note: the entry point is the address of the first instruction of the client program, not that of a procedure descriptor.

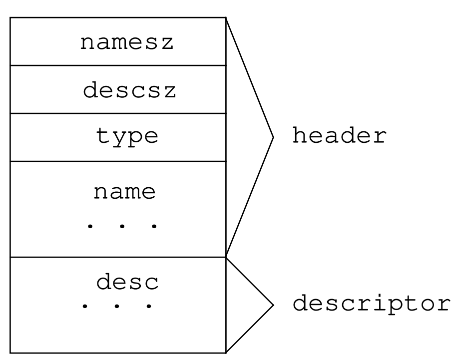
10.4.1. ELF-Format

This section defines how Open Firmware recognizes and prepares to execute an ELF-Format Program.

10.4.1.1. ELF Note Section

Part of the process of loading a client program involves verifying its environmental requirements (e.g., endian-ness and address translation mode) against the current firmware configuration. The client’s endian-ness can be directly determined by examining the ELF EI-DATA value; ELFDATA2LSB (1) implies Little-Endian while ELFDATA2MSB (2) implies Big-Endian. However, the other client requirements (e.g., address translation mode) are defined by means of an ELF Note Section (PT_NOTE), pointed to by the program header. The following describes the format of the Note Section for a client program file.

As defined by [19], an ELF file can be "annotated" by means of Note Sections within the executable file. A Note Section contains a "header" followed by a (possibly null) "descriptor", as follows:



Note: the endian format of the values corresponds to the endian-ness specified by the `EI-DATA` field of the file.

The format of a Note Section header can be described by an CHRP Open Firmware `struct` as:

```
struct   \ Note Section header for Open Firmware
/L   field    ns.namesz     \ length of ns.name, including NULL
/L   field    ns.descrsz
/L   field    ns.type
0    field    ns.name       \ NULL-terminated, /L padded
```

The `ns.name` field of the PowerPC Open Firmware Note Section *shall* be `"PowerPC"`; the `ns.type` field n *shall* be `0x1275.`

Following the Note Section header is a descriptor (`desc`); the length (in bytes) of the descriptor is specified by a word in the Note Section's header (`descsz`). The interpretation of the descriptor depends upon the kind of Note Section in which it is contained. For Open Firmware, the format of the Note Section's descriptor can be described by an Open Firmware `struct`, as follows:

```
struct   \ Note Section descriptor for CHRP Open Firmware
/L   field    ns.real-mode
/L   field    ns.real-base
/L   field    ns.real-size
/L   field    ns.virt-base
/L   field    ns.virt-size
/L   field    ns.load-base
```

If the **`ns.load-base`** value is not -1, then that value is compared against the current value of the **`load-base`** configuration variable. If they are equal no further action is taken. If they are not equal then the **`load-base`** configuration variable is set to the value of **`ns.load-base`** and the system is rebooted.

10.4.1.2. Recognizing ELF-Format Programs

The **`init-program`** *shall* recognize client program images that conform to all the requirements listed below as "ELF-format" programs.

In the description below, field names refer to fields within the ELF "file header" structure, which is assumed to begin at load-base, and offsets are relative to the beginning of that structure. Multi-byte numerical fields are interpreted according to the endianess specified by the "data" field at offset 5.

- a) The "e_ident" field (at offset 0) contains the string "\7fELF", where '\7f' is a byte whose value is (hex) 7f. This indicates the beginning of an ELF file header.
- b) The "EI_CLASS" field (at offset 4) contains the value 1. This indicates the 32-bit variant of the ELF format.
- c) The "e_type" field (at offset 16) contains the value 2. This indicates that the ELF image is executable.
- d) The "e_machine" field (at offset 18) contains the value 20. This indicates that the ELF image is for the PowerPC instruction set.
- e) The "e_version" field (at offset 20) contains the value 1.
- f) The "e_flags" field (at offset 36) contains the value 0.

10.4.1.3. Preparing ELF-Format Programs for Execution

Upon recognition of the client program image at load-base as an ELF-format program, init-program *shall* prepare the program for execution by performing the following sequence of steps.

In the description below, the fields mentioned by name are within ELF "program header" structures, unless specified otherwise.

- a) Search for an ELF "note" section of type "1275" as defined in the section "ELF Note Section". If one is found, and the values specified by its descriptor do not match the firmware's current operating mode, set the appropriate configuration variables to the values specified in the note section descriptor, and restart the firmware so that it will re-execute the **`boot`** command that resulted in the execution of **`init-program`**.
- b) Set the p_paddr field for each PT_LOAD segment equal to its p_vaddr field value if **`real-mode?`** is false and p_paddr is -1. This effectively maps these segments v=r.

- c) Allocate and map, if required, sufficient physical memory for all program segments of type PT_LOAD (i.e. whose "p_type" field contains the value 1) listed in the ELF image's program headers. Note that all PT_LOAD program segments that have a p_paddr value that matches their location in physical memory need not be moved, but the memory that they occupy must be claimed. This special case is added to allow large program images to be loaded without the 2x memory required to move the segments.
- d) Copy the program headers to a "safe" location to guard against the possibility of them being overwritten by the following steps.
- e) For each program segment of type "PT_LOAD":
    - 1) Copy, if required, the initialized portion of the program segment from its current location in the loaded image to the location given by the section's "p_paddr" field.
    - 2) Fill the rest of the segment with zero bytes (i.e., fill "p_filez - p_memsz" bytes beginning at the address "p_paddr + p_filesz").
    - 3) If **`real-mode?`** is false, then map the program segment to the virtual address specified by p_vaddr.
- f) Set the saved program state so that subsequent execution of **`"go"`** will begin execution at the address given by the "e_entry" field in the ELF file header. The e_entry field is a physical address if **`real-mode?`** is *true* and is a virtual address if **`real-mode?`** is *false*.

The implementation need not take precautions to ensure that the process of copying and zeroing program segments does not overwrite the portions of the load image that have not yet been copied. In order to guarantee correct copying, the value of the **`load-base`** configuration variable and the destination addresses of the various sections must be such that such overwriting does not occur. One sufficient condition is that the region of memory beginning at **`load-base`**, of size equal to the size of the loaded image, be disjoint from the regions of memory to which the program segments are copied and zero-filled. Another sufficient condition is to specify a **`load-base`** in the Notes Section (PT_NOTE) that ensures that the PT_LOAD segments are loaded at the address required by their program headers and thus are not moved. There are other less-stringent sufficient conditions, especially for simple ELF images with a small number of program segments that are to be copied to contiguous regions.

An implementation *shall* permit the ELF image to contain other program segments in addition to those described above, but need not take any action beyond that defined above as a result of the presence of such other program segments.

An implementation *shall* ignore all ELF sections. ELF sections are intended for binders, not loaders. Note that the CHRP ELF Note Section is actual an ELF segment of type PT_NOTE and thus the above does not apply to it.

10.5. Additional Client Interface Requirements

This section describes processor assist callbacks for real and virtual memory management and a test method service.

10.5.1. Client Interface Callbacks

This section describes callbacks for memory management. These callbacks are provided by the client.

10.5.1.1. Real-Mode Physical Memory Management Assist Callbacks

```
claim_mem
      IN:   [address] min_addr, [address] max_addr, size, align
      OUT:  throw-code, error,[address] real_addr
```

Allocate contiguous physical memory between min_addr and max_addr of size bytes (128KB max for an area in the 0 to 16MB address range), with align alignment. The alignment boundary is the smallest power of two greater than or equal to the value of align; an align value of 1 signifies one-byte alignment. A non-zero error code *shall* be returned if the mapping can not be performed. If error code is zero (i.e. allocation succeeded) the routine returns the real address (real_addr) of the physical memory block which was allocated for Open Firmware.

```
release_mem
      IN:   [address] phys, size
      OUT:  throw-code
```

Free size bytes of physical memory starting at real address phys, making that physical memory available for later use. That memory must have been previously allocated by claim_mem.

10.5.1.2. Virtual Address Translation Assist Callbacks

```
alloc_virt_mem
      IN:   size
      OUT:  throw-code, error,[address] virt_addr
```

Return the virtual address of a virtual memory area of size bytes aligned to a doubleword (8-byte) boundary. A non-zero error code *shall* be returned if the allocation can not be performed. If error code is zero (i.e. allocation succeeded) the routine returns the virtual address (virt_addr)of the memory block which was allocated.

```
free_virt_mem
      IN:   [address] virt_addr, size
      OUT:  throw-code
```

Free memory allocated by alloc_virt_mem. The values virt_addr and size must correspond with memory previously allocated by alloc_virt_mem.

```
claim_virt
      IN:   size, align
      OUT:  throw-code, error,[address] virt_addr
```

Allocate a memory area of size bytes and alignment align. The alignment boundary is the smallest power of two greater than or equal to the value of align; an align value of 1 signifies one-byte alignment. A non-zero error code *shall* be returned if the allocation can not be performed. If error code is zero (i.e. allocation succeeded) the routine returns the virtual address (virt_addr)of the memory block which was allocated.

```
release_virt
      IN:   [address] virt, size
      OUT:  throw-code
```

Free size bytes of virtual memory starting at virtual address virt, making that physical memory and the corresponding ranges of virtual address space available for later use. That memory must have been previously allocated by claim_virt.

10.5.2. Client Interface Services

Open Firmware *shall* provide the following *Client Interface Service*:

```
test-method
      IN:   [string] method, phandle
      OUT:  missing
```

Tests whether the package method named *method* exists in the package *phandle*. *Missing* is 0 if the method exists or -1 if the method does not exist.

11. Support Packages

This section describes the CHRP System binding specific requirements of Open Firmware support packages. These support packages are "`disk-label`" and "`tape-label`". For "`network`" and/or "`obp-tftp`" extensions, refer to [15]. These packages support the loading and executing of a client program. Another means of executing a Client Program is provided when an operating system ROM is a "bootable device"(Refer to Section 5.1.3. on page 25, as an example).

11.1. "disk-label" Support Package

The process of loading and executing a client program is described in two stages. The first stage determines what partition and/or file(if one exists) to read into memory. This is done by locating a partition and a file within the partition (if the partition supports a file system structure) from the boot device, usually by means of a name lookup within a directory contained within a disk "partition". The second stage examines the front portion (header) of the image for "well-known" program formats. When the format of the image has been determined, the loading is completed in a manner determined by that format.

The name of the partition (and, a file contained within the partition) can be explicitly specified by the user via the **`load`** or **`boot`** command, or can be implicitly specified by the value of the **`"boot-device"`** property of the "/options" node. The partition and filename are the ARGUMENTS portion of the final COMPONENT of the PATH_NAME, as described in section 4.3.1 of [1].

The syntax for explicit partition/filename specification is given in section 11.1.2. below where partition identifies the partition to be used and filename is the name of a file within that partition. If partition is omitted, the default partition (as determined by the partition format) is used. If filename is omitted, the default filename (i.e., the filename component of the "boot-device" path-name) is used.

11.1.1. Media Layout Format

This section describes the media layout formats of Client Program Images that the disk-label support package for a CHRP platform *shall* support; an implementation *may* support additional mechanisms, in an implementation-specific manner. The `"disk-label"` package for a CHRP platform *shall* support at least four(4) media layout types:

- FAT (FAT12 and FAT16 File System)
- FDISK (Partitions 4, 5, 6, 0x41 & 0x96)
- ISO-9660 (9660 File System)
- Mac OS (MAC Binary Image)

11.1.1.1. FDISK Partition Types

The following FDISK partition types *shall* be supported:

Partition Type 4: FAT 12 or FAT 16 File System

Partition Type 5: Extended Chained Partitions

Partition Type 6: Extended Partitions

Partition Type 0x41: Single program image

Partition Type 0x96: ISO 9660 File System

FDISK partition type 0 is a free partition. Partition type 0x82 is reserved and should not be used by CHRP Architecture.

11.1.2. Open Method Algorithm

The **`open`** method of the `"disk-label"` support package *shall* implement a disk partition recognition algorithm that supports at least the set of disk formats that are supported by the following algorithm. The following algorithm is intended to support raw (uninterpreted) disks, raw partitions of disks beginning with an FDISK partition map, and files on FAT and ISO-9660 file systems both within FDISK partitions and by themselves on disks without a partition map.

That **`open`** method *shall* accept an argument string (as returned by **`"my-args"`**) with the following syntax (according to the algorithm below), where brackets denote an optional component:

[partition][,[filename]]

If the argument string contains a comma, or if the argument string begins with a decimal digit, the partition component is deemed to be present. Note that the arguments above are not the client arguments with the boot command.

If the partition component is present, it selects the desired partition, where partition 0 refers to the entire disk, partition 1 refers to the first partition, partition 2 to the second, and so forth. If the partition component is absent and the disk has an FDISK or Mac partition map, the first "bootable" partition is used. If a “bootable” partition is not found, then fail in an implementation specific manner with an error.

If the filename component is present, it selects a particular file within the file system on the disk or partition thereof.

OPEN algorithm

```
Set D.SIZE to -1

If ARGUMENT$ is not the null string and the first character of ARGUMENT$ is
in the range '0'..'9' or is equal to ‘,’
     LEFT-PARSE ( ARGUMENT$ ) -> PARTITION$, FILENAME$

Else
     Set PARTITION$ to the null string
     Set FILENAME$ to ARGUMENT$

If PARTITION$ is not the null string

     If PARTITION$ is not a decimal number
          Return FALSE
     DECIMAL_STRING_TO_NUMBER( PARTITION$ ) -> PARTITION

     If PARTITION is 0
          GET_DISK_SIZE

     Else
          Read the first 512 bytes of the device into a buffer
          SELECT_EXPLICIT_PARTITION( PARTITION )
          If SELECT_EXPLICIT_PARTITION returned an error indication
               Return FALSE

Else  \ PARTITION$ is NULL
     Read the first 512 bytes of the device into a buffer
     SELECT_ACTIVE_PARTITION

     If SELECT_ACTIVE_PARTITION returned an error indication

          Return FALSE
\ At this point, D.OFFSET is set to the beginning of the selected partition
\ and D.SIZE is set to the size of that partition.  If the entire disk was
\ selected, D.OFFSET is 0 and D.SIZE is the size of the disk.

Call parent’s “seek” method with an argument of 0,0.

Return TRUE
```

CHECK_FOR_BPB procedure

```
If the first four(4) bytes are EBCDIC ‘IBMA’(hex character string C9C2D4C1),
```

then the sector does not contain a BPB.

If the 16-bit little-endian quantity beginning at buffer offset 510 is 0xAA55, and the 16-bit little-endian quantity beginning at buffer offset 11 (which is the BPB "bytes per sector" field) is either 256, 512, or 1024, and the byte at offset 16 (the BPB "number of FATs" field is either 1 or 2, the sector is deemed to contain a BPB. Otherwise, the sector does not contain a BPB.

CHECK_FOR_ISO_9660 procedure

Read 512-byte sector 64 (the beginning of logical 2048-byte sector 16) into a buffer.

If the byte at offset 0 contains the binary number "1", and the 5 bytes beginning at offset 1 contains the text string "CD001", the partition or raw disk is deemed to contain an ISO 9660 file system. Otherwise, the partition or raw disk is deemed not to contain an ISO 9660 file system.

CHECK_FOR_FDISK procedure

If the buffer does not contain an FDisk partition map signature of "AA55" as a 16-bit little-endian number beginning at buffer offset 510, the buffer is deemed not to contain an FDISK partition map.

If none of the partition type code field (the bytes at buffer offsets 0x1C2, 0x1D2, 0x1E2, and 0x1F2) contains a recognizable partition type code (4, 5, 6, 0x41, 0x96, or other types that may be recognized by the implementation), the buffer is deemed not to contain an FDISK partition map.

Otherwise, the buffer is deemed to contain an FDISK partition map.

The implementation may, at its option, apply additional validity tests to the partition map information.

CHECK_FOR_MAC_DISK procedure

If the first(i.e., at the lowest offset) two bytes in the buffer contains the 16-bit big-endian signature 0x4552, then the disk is deemed to be a Mac partitioned disk. Otherwise, the partition or raw disk is deemed not to be a Mac partitioned disk.

Note: Subsequent 512 byte sectors will contain Mac partition map entries, each of which begins with the 16-bit big-endian signature 0x504D. Each such partition map entry contains a field (V) indicating the total number of partition entries in the map.

INTERPOSE_BY_TYPE procedure

```
If FILENAME$ is not the null string

     If PARTITION-TYPE is 0x96

          INTERPOSE (ISO-9660 File System package, FILENAME$)
```

```
            Else

                 If PARTITION-TYPE is FAT,

                      INTERPOSE (FAT File System package, FILENAME$)
```

SELECT_ACTIVE_PARTITION(PARTITION) procedure

```
   CHECK_FOR_BPB

        If the buffer contains a BPB

             Set OFFSET to 0
             Set D.SIZE to the maximum size of the disk in bytes, as
             indicated by the information in the BIOS Parameter Block

        If FILENAME$ is not the null string

             INTERPOSE (FAT File System package, FILENAME$)

        Return OKAY

   CHECK_FOR_FDISK

        If the buffer contains an FDISK partition map

             Search the FDisk partition map, reading new 512-byte sectors into
             the buffer if necessary to "chain" to extended partition entries
             (i.e. ones whose type byte at offset 4 contains "5") for the
             first(i.e., at the lowest offset)partition entry whose "bootable"
             field (at offset 0 in the partition entry) contains 0x80.

             If a "bootable" partition was found:

                  Set PARTITION-TYPE to that entry's "type" field (the byte at
                  offset 4)

                  Set D.OFFSET to the byte offset from the beginning of the disk
                  of the beginning of the partition denoted by that entry.

                  Set D.SIZE to the size of the partition denoted by that entry.

                  INTERPOSE_BY_TYPE

                  Return OKAY

        \ (If this point is reached, no partition was marked "bootable")

        Search the FDisk partition map beginning in 512-byte sector
        0,reading new 512-byte sectors into the buffer if necessary to
        "chain" to extended partition entries, for the first partition(i.e., at
        the lowest offset) entry whose "type" byte is neither 0 nor 5 (5 is the
        type code that indicates a "chained" extended partition entry).

        If one is found:

             Set PARTITION-TYPE to that entry's "type" field (the byte at offset
             4)
```

```
          Set D.OFFSET to the byte offset from the beginning of the disk of the
          beginning of the partition denoted by that entry.

          Set D.SIZE to the size of the partition in bytes denoted by that
          entry.

          INTERPOSE_BY_TYPE

          Return OKAY

     \ (If this point is reached, the partition map did not contain any valid
        partition entries)

     Return ERROR

CHECK_FOR_ISO_9660

     If it is an ISO 9660 disk

          GET_DISK_SIZE

          If FILENAME$ is not the null string

               INTERPOSE (ISO-9660 File System package, FILENAME$)

     Return OKAY

CHECK_FOR_MAC_DISK

     If this is a Mac partitioned disk

          Search the Mac partition table for the first "bootable" partition.
          A partition is "bootable" when the pmPartStatus flags indicate that
          this is a valid, allocated, readable and bootable partition and the
          pmProcessor field contains "powerpc" (using case-insensitive
          matching).

          If a Mac "bootable" partition is found

               If FILENAME$ is “%BOOT”

                    If the Nth partition is marked bootable

                         Set D.OFFSET to the byte offset from the beginning of
                         the disk to the beginning of the boot area, as given
                         by the pmLgBootStart field.

                         Set D.SIZE to the size of the partition in bytes
                         denoted by pmBootSize.

                         Return OKAY

               Else

                    If the FILENAME$ is the null string

                         Set D.OFFSET to the byte offset of the “real”
                         partition data
```

```
                    Set D.SIZE to the size of the “real” partition data
               Else
                    INTERPOSE_BY_TYPE
          Return OKAY
     Else
          Return ERROR
(If this point is reached, no “bootable” partition was found)
Return ERROR
```

GET-DISK-SIZE procedure

```
Set OFFSET to 0
If the parent has a "#blocks" method
     Execute the parent's "#blocks" and "block-size" methods
          Set D.SIZE to the product of the numbers they returned
     Else
          Set D.SIZE to -1
```

SELECT_EXPLICIT_PARTITION procedure

```
CHECK_FOR_BPB
     If the buffer contains a BPB
          If PARTITION is 1
               Set OFFSET to 0
               Set D.SIZE to the maximum size of the disk in bytes, as
               indicated by the information in the BIOS Parameter Block
               If FILENAME$ is not the null string
                    INTERPOSE (FAT File System package, FILENAME$)
               Return OKAY
     Return ERROR
CHECK_FOR_FDISK
     If an FDisk partition map is found
          Search the FDisk partition map beginning in 512-byte sector 0,
          reading new 512-byte sectors into the buffer if necessary to "chain"
          to extended partition entries, for the Nth, where N is the value of
```

```
        PARTITION, partition entry whose "type" byte is neither 0 nor 5 (5
        is the type code that indicates a "chained" extended partition
        entry).

        If the Nth partition is found:

            Set PARTITION-TYPE to that entry's "type" field (the byte
            at offset 4)

            Set D.OFFSET to the byte offset from the beginning of the
            disk to the beginning of the partition denoted by that entry.

            Set D.SIZE to the size of the partition in bytes denoted by
            that entry.

            INTERPOSE_BY_TYPE

            Return OKAY

        Else

            Return ERROR

CHECK_FOR_MAC_DISK

    If this is a Mac partitioned disk

        Search the Mac partition map for the Nth partition, where N is
        the value of PARTITION.

        If the Nth partition is valid, allocated, and readable

            If FILENAME$ is %BOOT

                If the Nth partition is marked bootable

                    Set D.OFFSET to the byte offset from the beginning of
                    the disk to the beginning of the boot area, as given
                    by the pmLgBootStart field.

                    Set D.SIZE to the size of the partition in bytes
                    denoted by pmBootSize.

                    Return OKAY

                Else

                    Return ERROR

            If FILENAME$ is not the null string

                INTERPOSE_BY_TYPE

        Return OKAY

    \ (If this point is reached, this is an unrecognizable disk)

    Return ERROR
```

This algorithm can be used to locate the correct partition and/or file and/or load image from the specified device. The boot device is selected as described in 7.4.3.2 of [1]. A filename can be explicitly given as the arguments field of the *device-specifier* (i.e., the field following the ':' of the last path component). Other formats *may* be recognized in an implementation-specific manner.

11.2. **"tape"** Support

The tape device, for this version of the CHRP binding, *shall* be supported by the standard **"byte"** package as defined in [1].

11.3. **"network"** Support Package

The **"network"** Support Package *shall* adhere to the *Recommended Practice - obp-tftp Extension*[15] documentation functions and conventions.

11.4. Program-image formats.

Open Firmware must recognize a client program that is formatted as ELF [19] and PE [17]. Other formats *may* be handled in an implementation-specific manner. Reference [12] defines using FCode and Forth Program-Image Formats.

After locating the file, Open Firmware reads the image into memory at the location specified by the **load-base** Configuration Variable. Then, Open Firmware must perform the following procedure to prepare the image for execution.

init-program.

```
set restart? false

if the image is in ELF format(Refer to Section 10.4.1.2. on page 51)

     if the EI_DATA field does not match little-endian?

          set little-endian? appropriately.

          set restart? true

     locate the PowerPC Note Section

     if the Note Section's values do not match

          set Configuration Variables appropriately

          set restart? true

     if restart?

               restart the system, possibly by executing reset-all

     else

          move and/or relocate the ELF image
          (Refer to Section 10.4.1.3. on page 52).

          set GO's context with initial register values
```

```
else if the image is in PE format
     if little-endian? is false
          set little-endian? to true.
          restart the system, possibly by executing reset-all
     else
          move and/or relocate the PE image.
          set GO's context with initial register values
else if the image is FCode Image(hex characters F1,08)
     setup system and do subsequent go and perform a byte load of the FCode
     image

else if the image is Forth Code Source Image(ASCII characters \"<space>")
     setup system to evaluate Forth Source Image
else if the image is a bootinfo.txt file (i.e., begins with  "<CHRP-BOOT>")
     setup system to parse the bootinfo.txt file
else
     FAIL, in an implementation-specific manner.
```

12. Open Firmware Related NVRAM Usage

This section describes the NVRAM partitions for a CHRP platform. The platform Open Firmware defines the currently-defined NVRAM partitions and tags(see table below). Other partitions whose tags and usage are defined in the CHRP specification [3], Chapter 8, should be treated as RESERVED FOR FUTURE USE.

Table 12. Open Firmware Related NVRAM Partitions

Signature	Type	Ownership [1]	Number of partitions	Description [2]
0x50	Open Firm-ware	Firmware	1	For general OF usage
0x51	Reserved			Reserved
0x52	Hardware	Firmware	1 to n	For system information
0x53 - 0x5F	Reserved			Reserved
0x70	System	Global	1	For configuration variables
0x71	Configuration	Global	0 to n	For ISA and Power Management utilities data resource

Table 12. Open Firmware Related NVRAM Partitions

Signature	Type	Ownership [1]	Number of partitions	Description [2]
0x72	Error log	Global	0 to n	For errors during OF/RTAS
0x73	Reserved			Reserved
0x74 - 0x7E	Reserved			Reserved
0x7F	Free Space	Global	0 to n	Marks free space

Note 1: Ownership

Firmware - Read/Write of partition done only by firmware

Global - Read/Write of partition can be done by firmware and/or operating system

Note 2: Definitions

OF - Open Firmware

RTAS - Run Time Abstraction Services (OS environment)

Note 3: OS's each have their own private partition, using one of the signatures 0xA0 through 0xAF, Signature Type OS.

12.1. Firmware Partition

This partition reserved.

12.2. Multiboot Partition

This partition is reserved.

12.3. Hardware Partition

This partition is used for platform specific operations; i.e., like storage of a vendor Vital Product Data(VPD).

12.4. System Partition

The System Partition contains information that is accessible to both Open Firmware and operating systems. The contents of this partition are represented in the Open Firmware device tree as properties (i.e., (*name*, *value*) pairs) in the **`/options`** node. While Open Firmware is available, the operating system can alter the contents of these properties by using the **`setprop`** client interface service. When Open Firmware is no longer available, the operating system can alter the contents of the System Partition itself, following the rules below for the formats of the *name* and *value*. Information is stored in the System Partition as a sequence of (*name*, *value*) pairs in the following format:

name = *value*

where *name* follows the rules defined in Section 12.4.1 and *value* follows the rules defined in Section 12.4.2. The end of the sequence of pairs is denoted by a null (0x00) byte.

12.4.1. Name

Since the data in the System Partition is an external representation of properties of the **`/option`** node, the name component must follow the rules for *property names* as defined by Section 3.2.2.1.1 Property names of [1]; i.e., a string of 1-31 printable characters containing no uppercase characters or the characters `'/'`, `'\'`, `':'`, `'['`, `']'` or `'@'`. In addition to these rules, a naming convention is required for operating system specific names to avoid name conflicts. Each such *name* must begin with the OS vendor's OUI followed by a `','`; e.g., `aapl,`*`xxx`* or `ibm,`*`xxx`*. This introduces separate name spaces for each vendor in which it manages its own naming conventions.

12.4.2. Value

The value component of System Partition data can contain an arbitrary number of bytes in the range `0x01-0xFF`, terminated by a null (0x00) byte. Bytes in the range `0x01-0xFE` represent themselves. In order to allow arbitrary byte data to be represented, an encoding is used to represent strings of `0x00` or `0xFF` bytes. This encoding uses the `0xFF` byte as an escape, indicating that the following byte is encoded as:

`bnnnnnnn`

where **`b`**, the most-significant bit, is **`0`** to represent a sequence of `0x00` bytes or **`1`** to represent a sequence of `0xFF` bytes. **`nnnnnnn`**, the least-significant 7 bits, is a binary number (in the range `0x01-0x7F`) that represents the number of repetitions of `0x00` or `0xFF`.

12.4.3. Open Firmware Configuration Variables

Open Firmware configuration variables control the operation of Open Firmware. In addition to the standard configuration variables defined in [1], other configuration variables are defined by the PowerPC Processor binding [5] and by this binding. While such variables are stored in the System Partition as described above, they have additional rules placed on the format of the value component. Each configuration variable is also represented by a user interface word (of the same name) that returns stack value(s) when that word is evaluated. Each also has a platform defined default value; the absence of a configuration variable in the System Partition indicates that the value is set to its default value. The format of the external representation of configuration variables, and their stack representation, is defined by Section 7.4.4.1 Configuration Variables of [1]; the format depends upon the data type of the configuration variable. Whereas the internal storage format is not defined by [1], this binding specifies them as described below. The names of configuration variables are followed by the reference in which they are defined (where [*] refers to this binding).

12.4.3.1. Boolean Configuration Variables

The value of a boolean configuration variable is represented in the System Partition as the string `"true"` or `"false"`. The following configuration variables are of type boolean:

`auto-boot?` [1]

`diag-switch?` [1]

`fcode-debug?` [1]

`oem-banner?` [1]

`oem-logo?` [1]

`use-nvramrc?` [1]

`little-endian?` [5]

`real-mode?` [5]

`menu?` [*]

12.4.3.2. Integer Configuration Variables

The value of an integer configuration variable is represented in the System Partition as a decimal number or a hexidecimal number preceded by `"0x"`. The following configuration variables are of type integer:

`screen-#columns` [1]

`screen-#rows` [1]

`security-#badlogins` [1]

`security-mode` [1]

selftest-#megs [1]

real-base [5]

real-size [5]

virt-base [5]

virt-size [5]

load-base [5]

12.4.3.3. String Configuration Variables

The value of a string configuration variable is represented in the System Partition as the characters of the string. Where multiple “lines” of text are represented (e.g., in the nvramrc script), each line is terminated by a carriage-return (0x0D), a linefeed (0x0A), or carriage-return, linefeed sequence (0x0D,0x0A). The following configuration variables are of type string:

boot-command [1]

boot-device [1]

boot-file [1]

diag-device [1]

diag-file [1]

input-device [1]

nvramrc [1]

oem-banner [1]

output-device [1]

security-password [1]

bootinfo-nnnnn [*]

reboot-command [*]

12.4.3.4. Byte Configuration Variables

The value of a bytes configuration variable is represented by an arbitrary number of bytes, using the encoding escape for values of 0x00 and 0xFF. The following configuration variables are of type bytes:

oem-logo [1]

12.4.4. nvramrc Script

A CHRP Platform will implement the function **nvramrc** script as defined in [1] to allow an OS to change variable settings or initialize needed environment variables.

In addition, it is recommended for a CHRP Platform that has a hardware configuration with multiple SCSI Controllers on the same SCSI Bus to assign SCSI ID’s to avoid possible bus address conflicts. The mechanism to do this is a **nvramrc** script using a *multi-initiator* script defined below.

12.4.4.1. Multi-initiator Script

A CHRP Platform recommends the use of a **nvramrc** script to set the SCSI ID or *multi-initiator* for the next boot operation for a SCSI hardware configuration that is shared by two systems. That is, on SCSI buses where there are two or more SCSI Controller Adapters present on the same SCSI Bus. Below is a recommended script by which an OS can set the alternate SCSI Adapter ID to not conflict with the primary SCSI Controller.

nvramrc script for SCSI multi-initiator (This script is represented using new line characters <nl> to parse into a readable format):

dev device-specifier<nl>

nnn **encode-int " scsi-initiator-id"** property<nl>

device-end<nl><null>

where **'nnn'** is an integer specifying the SCSI ID. The <nl> is the ASCII new line character and <null> is the ASCII null character. Note that the formatting is for visual purposes.

Ground Rules:

- The **"scsi-initiator-id"** property defines the SCSI ID for that node (typically adapter). Absence of this property implies that SCSI ID 7 is in use by the adapter.
- This property can be assigned to any SCSI adapter node via the **nvramrc** script.
- Open Firmware processes the **nvramrc** script prior to performing any operations on the SCSI bus.
- The **nvramrc** script is contained in the NVRAM System Partition(0x71).

An example of a multi-initiator script follows:

Assume that the following are two SCSI adapter paths; `/pci@ff500000/pci3,1000@10` and `/pci@ff500000/pci3,1000@18`.

Then the following example sets the SCSI ID for /pci@ff500000/pci3,1000@10 to 6 and SCSI ID for /pci@ff500000/pci3,1000@8 to 7 at each boot before any SCSI operations.

```
nvramrc=dev /pci@ff500000/pci3,1000@10<nl>

6 encode-int" scsi-initiator-id" property<nl>

device-end<nl>

dev /pci@ff500000/pci3,1000@8<nl>

7 encode-int" scsi-initiator-id" property<cr>

device-end<nl>

<null>
```

12.5. Configuration Partition

The Configuration Partition(0x52) has two separate partitions with names of *isa-config* and *pm-config*(when power management is implemented by the platform). The ICU stores the ISA Resource information in the *isa-config* partition.

The device power management resources are stored in the *pm-config* partition by a CHRP platform Power Configuration Utility(PCU).

12.5.1. ISA PnP Configuration Resources

ISA legacy card resource data is stored in the *isa-config* partition of NVRAM as shown in Section 9.1.2. on page 45.

12.5.2. Device Power Management Configuration Resources

The format of power management information provided in the optional *pm-config* partition is defined in Section 9.2.3. on page 48.

12.5.3. Error Log Partition

This partition data format is defined in the CHRP Architecture for the possible error logs resulting from RTAS calls.

12.6. Global Partition

This section is reserved.

12.7. Free Space Partition

This partition signature is used to mark free NVRAM in a CHRP Platform

- END OF DOCUMENT -