PowerPC Reference Platform binding to:

IEEE Std 1275-1994

Standard for Boot

(Initialization, Configuration)

Firmware

Revision: 0.03 DRAFT

Date: August 5, 1996

1. Overview

This document specifies the application of *IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware, Core Practices and Requirements* to PowerPC Reference Platform Compliant computer systems, including practices for client program interface and data formats. An implementation of Open Firmware for a PowerPC Reference Platform Compliant system *shall* implement the core requirements as defined in [1], the PowerPC Processor-specific extensions described in [2] and the PowerPC Reference Platform specific extensions described in this binding.

This document defines the binding to PowerPC platforms that use 32-bit addressing. Since the minimum cell size of Open Firmware is 32 bits, only one cell is necessary to represent addresses of processor bus devices; hence, the value of "#address-cells" for / *shall* be 1.

> **Note: 64-bit platforms can follow the specifications contained herein as long as all addresses relevant to Open Firmware are kept within the first 4GB; i.e., the upper 32 bits of 64-bit addresses are assumed to contain 0s.**

2. References and Terms

2.1. References

This 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 When the following standards are superseded by an approved revision, the revision *shall* apply.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] *PowerPC processor binding to: IEEE Std 1275-1994, Standard for Boot (Initialization, Configuration) Firmware.*

[3] *System V Application Binary Interface*, published by UNIX System Laboratories. This document describes the generic architecture of the ELF (Executable and Linking Format) object file format.

[4] *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.

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

[6] *Bootstrap Protocol*, Internet RFC 951; see also RFC 1532.

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

[8] *System V Application Binary Interface, PowerPC Processor Supplement*, Sunsoft. This document defines the PowerPC specific ABI for System V and also gives details on the PowerPC ELF format.

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

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

[11] *Open Firmware Recommended Practice: Graphics Extension.*

[12] *Common Hardware Reference Platform Binding.*

2.2. Terms

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

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

ELF: Executable and Linking Format. A binary object file format defined by [3][8] 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 [4].

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

PE: Portable Executable. A binary object file format defined by [5]; this format is used by Microsoft's NT operating system.

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.

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.

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. Endian-ness Support

An implementation of PowerPC Reference Platform Open Firmware must 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.

3.1. 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 section 7.1.1.. When Open Firmware is started, Open Firmware must 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 system, including the processor(s). Open Firmware *shall* perform whatever steps are required to enable the system 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 system to be reconfigured until the system 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.**

4. Packages

This section describes the PowerPC Reference Platform-specific requirements of Open Firmware packages.

4.1. "**disk-label**" Support Package

This section describes the partition formats and the formats of client program images that the `disk-label` support package for PowerPC *shall* support; an implementation *may* support additional mechanisms, in an implementation-specific manner.

The process of loading and executing a client program is described in two stages. The first stage determines what file to read into memory. This is done by locating a file 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 file (and, the partition in which it is contained) 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 filename is the `ARGUMENTS` portion of the final `COMPONENT` of the `PATH_NAME`, as described in section 4.3.1 of [1].

The syntax for explicit filename specification is as follows:

```
[n][,filename]
```

where `n` is the partition number to be used and `filename` is the name of a file within that partition. If `n` 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. Partition number 0 refers to the entire device; this definition is independent of the existence of the device partition information.

4.1.1. Partition formats

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.

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 partition map, the first "bootable" partition is used, otherwise the first non-null partition is used.

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 the first character of ARGUMENT$ is in the range '0'..'9'
        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$ cannot be parsed as a decimal number
             Return FALSE
     DECIMAL_STRING_TO_NUMBER( PARTITION$ ) -> PARTITION
     If PARTITION is 0
             Set OFFSET to 0
             If the parent device has a "#blocks" method
                     Execute the parent's "#blocks" and "block-size"
                     methods, and set D.SIZE to the product of the
                     two result values.
     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.

If FILENAME$ is not NULL

     If the partition contains an ISO 9660 file system

             Interpose the ISO 9660 file system handler, with FILENAME$
             as its argument

     Else If the partition contains a FAT file system

             Interpose the FAT file system handler, with FILENAME$
             as its argument

     Else If the implementation recognizes the file system type and
     has a handler for that file system

             Interpose the handler for that file system, with FILENAME$
             as its argument

     Else
             Return FALSE

Return TRUE


CHECK_FOR_BPB procedure

     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.


GET_DISK_SIZE procedure

    If the parent device has a "#blocks" method

            Execute the parent's "#blocks" and "block-size"
            methods, and set D.SIZE to the product of the
            two result values.


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 the partition map area (the bytes of the buffer between offsets
    0x1ae and 0x1fd inclusive) contains all zero bytes, the buffer is
    deemed not to contain an FDISK partition map.

    If none of the partition type codes (the bytes at buffer offsets
    0x1b2, 0x1c2, 0x1d2, and 0x1e2) 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 tests to
    determine whether or not the buffer contains a valid partition map.

INTERPOSE_BY_TYPE procedure

    If FILENAME$ is not the null string

            If PARTITION-TYPE is 0x96
                    INTERPOSE (ISO-9660 File System package, FILENAME$)
            Else
                    INTERPOSE (FAT File System package, FILENAME$)

SELECT_ACTIVE_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 does not contain an FDISK partition map

        CHECK_FOR_ISO_9660
        If it is an ISO 9660 disk

                Set OFFSET to 0

                GET_DISK_SIZE

                If FILENAME$ is not the null string

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


                Return OKAY

        Return ERROR

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 contain "5")
for the first partition entry whose "bootable" field (at
offset 0 in the partition entry) contains 0x80.
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 denoted by
        that entry.

        INTERPOSE_BY_TYPE

        Return OKAY
```

Note: An implementation is required to support only the "logical sector number" format of FDISK partition map entries. It may, but is not required to, support the "head/cylinder/sector" format.

```
\ 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
        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

        \ The partition map did not contain any valid partition entries

        Return ERROR

   GET-DISK-SIZE procedure

        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(PARTITION) procedure

        If the buffer contains a BPB

                If PARTITION is 1

                        Set OFFSET to 0

                        GET-DISK-SIZE

                        If FILENAME$ is not the null string

                             INTERPOSE (FAT File System package, FILENAME$)

                        Return OKAY

                Else
                        Return ERROR
        Else
              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 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

        Else

                Return ERROR
```

Note: Some vendors wish to supply a CD-ROM that contains both an ISO-9660 file system, and also a separate boot image in another format. One way do so, that will work with the algorithm described above, is as follows:

Put the ISO-9660 file system on the CD-ROM in its usual location (i.e. beginning at 2048-byte sector number 16).

Put the boot image, in one of the defined partition formats (e.g. as a FAT file system or a "41" partition) on the CD-ROM in some area beyond that which is used by the ISO-9660 file system.

Put an FDISK partition map in the first 512 bytes of the CD-ROM (i.e. in the first quarter of the first 2048-byte sector). That partition map should contain:

a) An entry describing the boot image partition. The logical sector numbers in that entry should be in units of 512 bytes. That entry should be marked "bootable" (i.e. the first byte of the entry should be 0x80).

and

b) An entry describing the ISO-9660 file system. The logical sector numbers in that entry should be in units of 512 bytes. The beginning sector number field in that entry should contain 0, thus "including" the FDISK partition map within the ISO-9660 partition.

Although the above algorithm works independent of the device type, the following formats are strongly recommended for devices for portability.

Floppy Disk

1.44/2.88 MB, MFM floppy disks should be in FAT-12 format, as described in [4].

Hard Disk

Hard Disks should have an FDISK partition map, as described in [4].

Note: since the bootsector is used to contain boot program for floppies and the FDISK partition map for hard disks, the "`disk-label`" package must use the value of the `bsMedia` byte (located at offset `15h`) to determine whether a partition map is present. If the value is `0F8h`, it indicates a hard disk and a partition map should be present in the bootsector; any other value indicates a floppy disk.

CD-ROM

CD-ROMs should be formatted according to ISO-9660, as described in [7].

4.1.2. Program-image formats.

Open Firmware must recognize a client program that is formatted as either ELF [3][8] or PE [5]. PE format support is provided only for booting NT; all other clients *shall* use ELF. Other formats *may* be handled in an implementation-specific manner.

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.

```
   examine the header of the image.
   set restart? false
   if the image is in ELF format
        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 descriptor is not correct
             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.
   else
   if the file 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.
   else
        FAIL, in an implementation -specific manner.
```

4.2. "obp-tftp" Support Package

The "obp-tftp" Support Package of an Open Firmware implementation (used to **load** from "network" devices) *shall* support the BOOTP protocol, as described in [6] and *shall* not require the server to support any vendor extensions.

5. Properties

This section describes the standard properties of PowerPC Reference Platform Open Firmware implementation.

5.1. Root node properties

The following properties of the root node ("/") shall be creates by an Open Firmware implementation.

> **Note: The root node typically corresponds to the common processor bus in a PowerPC system**

"#address-cells"

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.

"clock-frequency"

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

"system-id"

Standard property, encoded as with **encode-string**, that contains the identification of the computer system. This string shall be unique across all systems and all manufacturers. If the system-id begins with "0", it shall be of the form "0nnnnnnmmmmmm" where nnnnnn is a sequence of 6 uppercase hexadecimal digits representing a 24-bit Organizationally Unique Identifier (OUI) assigned by the IEEE Registration Authority Committee, 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 systems with built-in ethernet or other IEEE 802-style interfaces, the 6-byte MAC address assigned to that interface meets the requirements and should be used as the system-id.

6. Halt/Reboot

The "halt procedure" feature defined in earlier versions of this binding has been deleted in this version in favor of a similar service provided by the "RTAS" services defined by the CHRP binding[12]. PR*P firmware that wishes to provide a reboot service that will work after an operating system has rendered the Open Firmware client interface inoperable should implement the RTAS "reboot" service.

7. Client Program Requirements

7.1. 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 [3][8], and extended by section 7.1.1., or PE (Portable Executable) as defined by [5]. The standard ELF format contains explicit indication as to the program's execution modes (e.g., 32- or 64-bit, Big- or Little-Endian); this version of the specification only deals with the 32-bit version (i.e., `ELFCLASS32`).

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

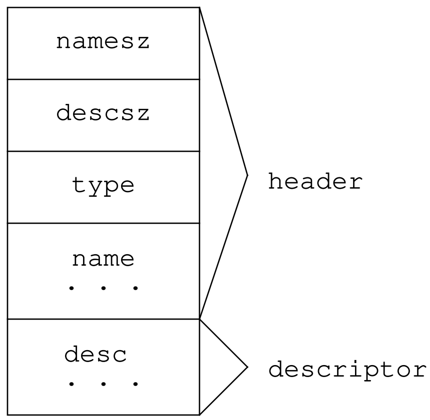
A standard *client program* shall be statically linked, requiring no relocation of the image. The program's entry point (`e_entry`) *shall* contain the address of the first PowerPC instruction of the client program. It is the responsibility of the client program to establish the appropriate value of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

7.1.1. 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 (SHT_NOTE). The following describes the format of the Note Section for a client program file.

As defined by [3], 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 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 *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
/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
```

7.1.2. Handling ELF-format Client Programs

7.1.2.1. Recognizing ELF-format Programs

"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 endianness specified by the "data" field at offset 5.

a) The "`ident`" field (at of fset 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 "`class`" field (at of fset 4) contains the value 1. This indicates the 32-bit variant of the ELF format.

c) The "`type`" field (at offset 16) contains the value 2. This indicates that the ELF image is executable.

d) The "`machine`" field (at of fset 18) contains the value 20. This indicates that the ELF image is for the PowerPC instruction set.

7.1.2.2. 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 Section 7.1.1. on page 11. 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) Allocate and map sufficient physical memory for all the program sections of type "`PT_LOAD`" (i.e. whose "`p_type`" field contains the value 1) listed in the ELF image's program headers.

c) Copy the program headers to a "safe" location to guard against the possibility of them being overwritten by the following steps.

d) For each program section of type "`PT_LOAD`":

   1. Copy the initialized portion of the program section from its current location in the loaded image to the location given by the section's "`p_vaddr`" field.

   2. Fill the rest of the copied section with zero bytes (i.e. fill "`p_filez - p_memsz`" bytes beginning at the address "`p_vaddr + p_filesz`").

e) Set the saved program state so that subsequent execution of "**`go`**" will begin execution at the virtual address given by the "`e_entry`"field in the ELF file header .

The implementation need not take precautions to ensure that the process of copying and zeroing program sections 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 sections are copied and zero-filled. There are other less-stringent sufficient conditions, especially for simple ELF images with a small number of program sections that are to be copied to contiguous regions.

An example of another sufficient condition would be as follows:

An ELF client program consists of two segments: a text segment (read/execute) and a data segment (read/write/execute). Only the data segment may have a memory size greater than the file size (implied "bss"). Further , to ease the problem of mapping the image for Virtual mode, the page offsets within memory must equal the page offsets within the file. The definition of "page offset" is those bits of an address which are below the granularity of memory allocation or mapping. (i.e. page size of 4K implies page of fset mask of 4K - 1)

An implementation shall permit the ELF image to contain other sections 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 sections.

8. Extensions for PowerPC Reference Platform systems

This section describes the properties, methods, and device subtrees that are applicable to devices required by the PowerPC Reference Platform architecture. It is strongly recommended that other platforms follow these definitions for the corresponding devices.

8.1. Display devices

Display device packages (i.e.., `device_type = "display"`) for PowerPC Reference Platform systems should include in their implementation all the properties and methods called for in [10] and [11].

8.2. Device support

Open Firmware implementations for PowerPC Reference Platform systems must implement those device types from the IEEE 1275 Device Support Extensions document [9] that are appropriate to the hardware present.

8.3. Conventions for devices on ISA and SCSI buses

This section defines the naming and device type conventions for typical devices on ISA and SCSI buses. The following lists are the values of the "name" and "device-type" properties of the devices on an ISA bus:

```
name                device_type
8042
     kbd            "keyboard"
     mouse          "mouse"
fdc                 "fdc"
     disk           "block"
     tape           "byte"
com                 "serial"
timer               "timer"
lpt                 "parallel"
ide                 "ide"
     disk           "block"
     cdrom          "block"
nvram               "nvram"
rtc                 "rtc"
```

Note: The "kbd" and "mouse" names are indented to show that they are the child nodes of the 8042 node.

Some systems use an I/O controller, often called a super I/O chip, which provides control functions of multiple I/O devices. When a system uses a super I/O chip, a device node representing the super I/O chip itself need not exist. Instead, the device nodes of the devices attached to the super I/O chip may be direct children of the bus node representing the bus to which the super I/O chip is attached.

The following are the values of the name and device-type properties of the devices on SCSI bus:

```
name                device_type
scsi                "scsi-2"
     disk           "block"
     tape           "byte"
```

Note: SCSI controller is considered a bus device in the device tree for PowerPC Open Firmware. The "disk" and "tape" names are indented to show that they are the child nodes of the scsi node.

It is strongly recommended that the "compatible" property be implemented for ISA and SCSI bus devices to help operating systems find appropriate device drivers for these devices.

8.4. /aliases node properties

An implementation of Open Firmware for the PowerPC Reference Platform shall provide the following aliases under "`/aliases`" node if an applicable device exists:

```
disk
tape
```

```
cdrom
keyboard
screen
scsi
com1     The serial port at I/O address 0x3f8
com2     The serial port at I/O address 0x2f8
floppy   The primary floppy drive
net
```