Most simple device drivers will in fact act in both roles, but other configurations are possible. Vendors who offer a system with components different from those in the Reference Implementation must support this differentiation with a device driver. In simple cases where the device head and the device handler are in the same device driver, the system vendor would supply both with the system. In another case where the device head and device handler are in different device drivers, the system vendor would only need to match the interface to the device head and supply a device handler.

Documents useful to vendors needing to develop device drivers are as follows:

- *Kernel Extensions and Device Support Programming Concepts*, IBM document number SC23-2207
- *Writing a Device Driver for AIX Version 3.2*, IBM document number GG24-3629

E.10 Multiprocessing Model

| AIX Version 4.1.1 supports SMP. However, there is no multiprocessor support in AIX for systems based
| on the PowerPC Reference Implementation as defined in Section 6.0, "Reference Implementation."

E.11 Application Model

| AIX Version 4.1.1 will support the PowerOpen Environment (POE). This environment provides an application interface which is targeted to be platform and I/O independent. The goal of the POE Application Binary Interface (ABI) is to free software developers from having to consider platform-specific functions and I/O bus dependencies. The ABI is designed to help enable software vendors to produce shrink-wrapped software. The ABI contains the Applications Programming Interface (API), which provides access to operating system libraries and services.

The PowerOpen ABI defines a system interface for compiled application programs. The purpose of this approach is to standardize the binary interface for application programs or some other operating system that complies with the POE specification.

The PowerOpen ABI execution environment contains information that must be provided by a PowerOpen operating system and is available to a PowerOpen application program. The ABI defines a binary interface for application programs that are compiled and packaged for POE implementations on similar hardware architectures. The binary specification includes information specific to the PowerPC computer processor architecture.

An implementation conforming to this standard must meet the following criteria:

- The system shall support all the PowerOpen execution environments, interfaces and headers defined and listed within the ABI specification, which is part of the POE.
- The system may provide additional or enhanced interfaces, headers, and facilities not required by the ABI specification, provided that such additions or enhancements do not affect the behavior of an application that requires only the facilities described in the ABI specification.

Adherence to the PowerOpen ABI guarantees application portability to future versions of an ABI-conformant system and to future PowerPC architecture implementations. This portability is guaranteed at the following levels, depending on the application's origin, as follows:

ABI-conforming system A computer system that provides all the binary system interfaces for application programs described in the ABI specification and the PowerOpen API.

ABI-conforming program A program written to include only the following system routines:

Commands and other resources included in the ABI.

Programs compiled into an executable file that has the formats and characteristics specified for such files in the chapter on XCOFF in the ABI specification.

Programs whose behavior complies with the rules given in this ABI specification.

A program must not bind in the routines and data structures from the system shared libraries defined in the chapter on libraries in the ABI specification. A program cannot have the routines defined in the shared libraries statically bound into the program.

Binary compatibility Presents a load-and-go environment. Only the physical availability of the application is needed. Applications adhering to this level of compatibility can be moved across compliant systems. Note that the static linking of shared libraries cannot be guaranteed to work and must therefore be avoided.

E.12 Configuration Summary Table

Table 31 gives the minimum configuration information for each of the three AIX system configurations. This table specifies levels of hardware necessary to run the three AIX system configurations. In many cases upgrade options are possible. Some of those options and configuration alternatives were described in Section E.5, "Hardware Configuration Requirements."

Table 31 (Page 1 of 2). Hardware Requirements for AIX System Configurations

	System Component	Client	Developer	LAN Server
	Processor	PowerPC 601, 603, 604, or compatible processor		
	System Memory	16 MB	24 MB	32 MB
	System ROM	Standard	Standard	Standard
	Non-volatile RAM	Standard -- 4 KB	Standard -- 4 KB	Standard -- 4 KB
	L2 Cache Memory	Optional	Optional	Optional
	Hardfile	200 MB	400 MB	> = 400 MB
/	Floppy	Optional	Optional	Optional
/	CD-ROM*	Required	Required	Required
/	Alphanumeric Input Device	Required	Required	Required
/	Pointing Device	Required	Required	Optional
/	Audio	Standard	Standard	Standard
	Graphics	800x600	1024x768	ASCII Characters
	Real-Time Clock	Standard	Standard	Standard
/	Serial Ports	2 Required	2 Required	2 Required
	EIA/TIA-232E	AIX supports this serial interface		
	EIA-422	AIX does not support this serial interface		
/	Parallel Ports	1 Required	1 Required	1 Required

Table 31 (Page 2 of 2). Hardware Requirements for AIX System Configurations

System Component	Client	Developer	LAN Server
Compatibility Mode	AIX supports this parallel interface		
Extended Capabilities Port	AIX does not support this parallel interface		
Network Adaptors	Optional	Optional	Required
Ethernet	AIX supports Ethernet		
Token Ring	AIX supports Token Ring		
SCSI	1 Required	1 Required	1 Required
IDE	Not supported	Not supported	Not supported
PCI Bus	1 Required	1 Required	1 Required
ISA Bus	Required	Required	Required
PCMCIA	Not currently supported -- a vendor-supplied device driver and kernel extensions are required		
Tape Drive	Optional	Optional	Optional

Legend:

Entry	Definition
Standard	The hardware configuration in Section 2.0, “Hardware Configuration” requires this component.
Required	This component must be present in systems on which AIX will run.
Optional	This component is supported by AIX, but is not required.
*	Capability can be provided via a network.

Appendix F. Workplace OS

/ This appendix describes OS/2 for the PowerPC, which is the first member of the Workplace OS* family of
/ operating systems. OS/2 for the PowerPC runs on PowerPC Reference Platform-compliant machines.

F.1 Operating System Scope

/ Workplace OS defines a family of general-user operating systems which consist of the IBM microkernel, Per-
/ sonality Neutral Services, and multiple personalities. The first member of the Workplace OS family is OS/2
/ for the PowerPC, which runs on PowerPC Reference Platform-compliant machines. The available personal-
/ ities are OS/2* and MVM (DOS personality). The user sees this as a portable OS/2. This document
/ describes the first implementation of Workplace OS and thus the terms Workplace OS and OS/2 for the
/ PowerPC can be used interchangeably. Future members of the Workplace OS family may have slightly
/ different configurations and requirements. Future personalities include UNIX**.

F.2 Operating System Version

This Appendix describes the first release of Workplace OS. This version provides to users and applications the services that they are currently using in OS/2 2.1.

/ F.3 Operating System Environment

/ OS/2 for the PowerPC runs in Little-Endian mode. OS/2 for the PowerPC supports the ELF object
/ module format with Workplace OS extensions. The linkage convention is the common Workplace OS,
/ SunSoft, and embedded processor conventions based on GOT (Global Offset Table). Emulation capabilities
/ are available for 80486 ring 3, DOS 6.3, Windows 3.11, and WIN32s. OS/2 for PowerPC supports the FAT
/ (DOS), HPFS, and ISO 9660 file systems.

F.4 Operating System Configuration

Workplace OS is offered in a single scalable configuration. This configuration may be used as a client or developer workstation. It can also be used as a server through the addition of products such as the IBM LAN Server for Workplace OS.

F.5 Hardware Configuration Requirements

This section defines the minimum and recommended hardware configurations. In many cases performance improvements can be realized by configuring systems with larger or faster components. Because these configuration adjustments are application load dependent, no attempt has been made to recommend operationally tuned configurations.

The subsections below describe the configuration recommendations, which are summarized in Table 32.

F.5.1 Processor Subsystem

A PowerPC Reference Platform-compliant system must have a PowerPC compliant processor. Workplace
/ OS will support PowerPC 601, 603, 604 or compatible PowerPC processors running at supported frequencies. Performance will be better with higher frequencies and with PowerPC processors having performance-enhancing features such as larger internal cache.

F.5.2 Memory Subsystem

System Memory for a Workplace OS client workstation should be at least 8 MB (16 MB is recommended). For a developer workstation, System Memory should be 16 MB. Servers may require additional memory. Performance improvements can be realized with additional System Memory.

System ROM will not vary between configurations. The minimum size is determined by what a vendor needs to boot the machine into the state expected by the operating system as defined in Section 5.0, "Boot Process and Firmware."

Workplace OS does not require a cache external to the processor. The cache is recommended for developer and server configurations.

Workplace OS uses the Non-volatile Memory for configuration and global environment storage. This information is shared between multiple operating systems (or multiple versions of the same operating system) on the platform. Non-volatile Memory requirements are the same for all workstation configurations.

F.5.3 Storage Subsystems

The minimum hardfile size for a Workplace OS client workstation is 120 MB. A developer workstation requires at least 240 MB. The minimum size for a server depends upon its function, but should not be smaller than 320 MB.

/ A 1.44-MB 3.5-inch floppy drive is required for all workstations to allow information interchange. This
/ requirement can be met by having external or network access to such a floppy device. Workplace OS also supports a 2.88-MB 3.5-inch floppy drive.

A CD-ROM is required for all Workplace OS machines. The Workplace OS distribution media will be CD-ROM. An acceptable alternative to internal CD-ROM support is an external CD-ROM, or network access to a CD-ROM. All workstations benefit from high-access-speed CD-ROMs.

F.5.4 Human Interface Subsystem

The video system must be capable of showing 640x480x8. Higher resolutions and additional color depth are
/ recommended for clients and developers. When emulating DOS applications which require planar graphics
/ (e.g. games and Prodigy*), all configurations must have a VGA-compatible video system.

A keyboard and pointing device are required for all workstations.

/ A business audio device is required as part of the standard hardware configuration described in Section 2.0,
/ "Hardware Configuration."

F.5.5 Connectivity Subsystem

/ Workplace OS does not require a serial port, but a serial port is required as part of the standard hardware
/ configuration described in Section 2.0, "Hardware Configuration." Workplace OS supports EIA/TIA-232-E-compliant serial ports. Any configuration may contain one or more serial ports.

Workplace OS does not require a parallel port. Workplace OS supports 8-bit bidirectional Compatibility
/ Mode-compliant parallel ports. Initial versions of Workplace OS will support an ECP parallel port, but may
/ not exploit its performance advantages.

/ No network connection is required. Workplace OS does not provide network support, but this support may
/ be provided through add-on products.

F.5.6 Expansion Bus Interfaces

/ Workplace OS requires one and only one PCI bus.

Workplace OS supports PCMCIA, including socket services. A device driver must be included for non-standard devices.

Workplace OS supports multiple instances of SCSI interfaces.

/ Workplace OS has very complete support for IDE.

The ISA bus is optional.

/ F.5.7 Security and System Management

/ Workplace OS has no special requirements in this area.

F.6 Hardware Configuration Recommendations

The Workplace OS system which supports PowerPC Reference Platform-compliant systems will support additional equipment. In some cases, drivers will have to be supplied by either the system vendor or the device vendor.

Audio upgrades with better sound and MIDI support would be useful additions to facilitate HUMAN-CENTERED applications.

Alternative graphics adaptors may be supplied on PowerPC Reference Platform-compliant machines.
/ Workplace OS initially supports S3, Weitek, and Western Digital adaptors. Vendors would have to provide drivers for other adaptors.

/ Workplace OS requires VGA capabilities to emulate DOS programs which use VGA planar modes.

/ Tape drives are useful as backup devices and archive mechanisms. Workplace OS supports tape devices in
/ the same manner as it supports other peripheral devices, and will contain sample device drivers.

F.7 Boot Time Abstraction Requirements

Workplace OS uses boot devices defined by the firmware. Only those devices (e.g. video, keyboard, disk) known to the firmware participate in the boot process.

/ Hardware abstraction is provided in Workplace OS by Open Firmware or by a customized microkernel. A
/ shrink-wrapped Workplace OS can be used with a custom microkernel, but this combination will decrease
/ system's the ability to fully boot from CD-ROM.

F.8 Hardware Abstraction Layer

Workplace OS has been designed to be portable to multiple hardware platforms. The portable nature of Workplace OS is a result of the use of a microkernel as the foundation of the operating system.

The microkernel is a small, message-passing nucleus of systems software running in the most privileged state of the computer. It supports the rest of the operating system as a set of applications called servers. The microkernel encapsulates the processor-specific information and other hardware dependencies. The other operating system components (i.e. the servers) communicate with the microkernel through a rich set of interfaces. The interfaces shield the servers from the processor architecture and system structure.

The microkernel supports a basic set of system services: virtual memory management, tasks and threads, interprocess communications, I/O support and interrupt management, and processor services. Higher-level operating system services, such as file system support, device driver support and application interfaces, are supported by out-of-kernel independent servers. This structure ensures that system integrity and security is not compromised by either misbehaved applications or device drivers. The system can scale to support diverse hardware and software configurations.

In the Workplace OS architecture, traditional application programming interfaces are supported by servers called personalities. Workplace OS for PowerPC Reference Platform-compliant systems provides the OS/2 personality and DOS/Windows** 3.1 support in the first release.

Figure 61 shows the Workplace OS structure. The underlying hardware is managed by the microkernel. Device drivers, the file system, and the OS/2 personality are user-level processes (not privileged). Applications are written to the interfaces exported by the OS/2 personality. This structure allows Workplace OS to easily support multiple hardware variants while ensuring portability of the operating system and user applications.

F.9 Device Driver Model

There will be a separate Device Driver Development Kit (DDK) available from IBM. It contains all the documentation and sample code needed to develop device drivers.

In addition, DOS device drivers may be used to access devices from a virtual DOS session.

F.10 Multiprocessing Model

The IBM microkernel, the Workplace personalities, and other Workplace OS components are designed to
/ support SMP. Workplace OS has set no specific limit on the number of processors in an SMP.

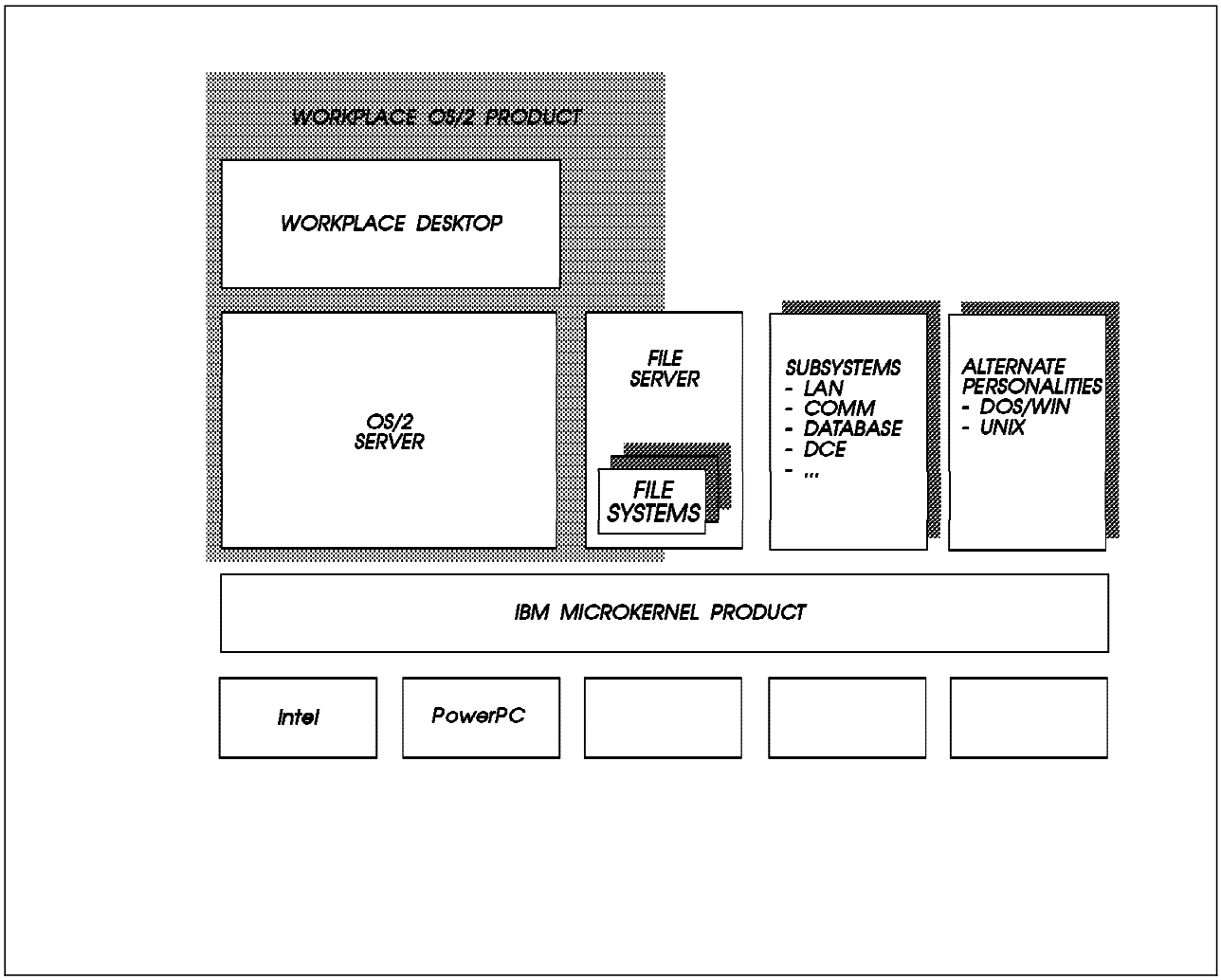

Figure 61. Workplace OS Structure

F.11 Application Model

Applications of Workplace OS/2 are written at several levels:

a) OS/2 32-bit PowerPC applications

b) DOS and Windows 16-bit Intel emulation applications

Thirty-two-bit applications written for OS/2 2.1 are source-compatible with the Workplace OS/2 personality. Such applications can make full use of the machine graphics capabilities using Presentation Manager.

Existing Intel x86-architecture DOS- and Windows-based applications can be run in DOS mode using an instruction set emulator (ISE). Workplace OS also provides a full emulation of both DOS and Windows.

Shared services and subsystems such as communication, database, and file servers can be written as Personality Neutral Services. Such subsystems can directly access the capabilities of the IBM microkernel and are available to applications as well as other subsystems.

F.12 Configuration Summary Table

Table 32 gives the minimum configuration information for three Workplace OS system configurations. This table specifies levels of hardware necessary to run these configurations. In many cases upgrade options are possible. Some of those options and configuration alternatives were described in Section F.5, "Hardware Configuration Requirements." As additional data for other personalities becomes available this table will be updated with similar data for each personality.

Note: The information in this table does not include the impact of DOS emulation software.

Table 32 (Page 1 of 2). Hardware Requirements for Workplace OS System Configurations

	System Component	**Client**	**Developer**	**Server**
/	Processor	PowerPC 601, 603, 604, or compatible processor		
	System Memory	8 MB	16 MB	> = 16 MB
	System ROM	Standard	Standard	Standard
	Non-volatile RAM	Standard 4 KB	Standard 4 KB	Standard 4 KB
	L2 Cache Memory	Optional	Optional	Optional
/	Hardfile	120 MB	240 MB	> = 320 MB
/	Floppy (1.44 MB)*	Required	Required	Required
/	CD-ROM*	Required	Required	Required
/	Keyboard	Required	Required	Required
/	Pointing Device	Required	Required	Required
/	Audio	Standard	Standard	Standard
	Graphics	640x480	640x480	640x480
	Real-Time Clock	Standard	Standard	Standard
/	Serial Ports	Standard	Standard	Standard
	EIA/TIA-232-E	Workplace OS supports this serial interface		
	EIA-422-A	Workplace OS does not support this serial interface		
/	Parallel Ports	Optional	Optional	Optional
	Compatibility Mode	Workplace OS supports this parallel interface		
/	Extended Capabilities Port	Workplace OS supports this parallel interface		
/	Network Adaptors	Optional	Optional	Required
/	SCSI	Optional	Optional	Optional
	IDE	Optional	Optional	Optional
	PCI Bus	1 Required	1 Required	1 Required
	ISA Bus	Optional	Optional	Optional
	PCMCIA	Optional	Optional	Optional

Table 32 (Page 2 of 2). Hardware Requirements for Workplace OS System Configurations

System Component	Client	Developer	Server
Tape Drive	Optional. A vendor-supplied device driver is required.		

Legend:

Entry	Definition
Standard	The hardware configuration in Section 2.0, "Hardware Configuration," requires this component.
Required	This component must be present in systems on which Workplace OS will run.
Optional	This component is supported by Workplace OS, but is not required.
*	Capability can be provided via a network.

Appendix G. Solaris

/ This appendix describes the Solaris** operating environment which runs on PowerPC Reference Platform-
/ compliant machines.

/ G.1 Operating System Scope

/ The Solaris operating environment is SunSoft's next-generation distributed computing solution. A fully
/ compliant implementation of System V Release 4 UNIX (SVR4), Solaris provides access to the most pow-
/ erful and advanced solutions available, including an open, fully networked environment. Solaris also unites
/ the world's largest installed base of CISC and RISC hardware -- SPARC, x86, and PowerPC.

/ The SunOS** 5.x operating system is the foundation of Solaris. The innovative technology it contains --
/ including the industry's leading implementation of a multithreading operating system -- brings a new level of
/ performance to Solaris users. SunOS 5.x builds on SVR4 and extends it by introducing new features
/ including symmetric multiprocessing with a multithreaded kernel, real-time functionality, advanced interna-
/ tionalization, and increased security features.

/ ONC+**, Solaris' core networking technology, is one of the world's most widely used heterogeneous net-
/ working solutions. The ONC+ family of protocols and distributed services, including NFS**, is supported
/ by over 300 companies and has an installed base of over 3 million users.

/ Solaris features OpenWindows**, SunSoft's X11, network-based windows system. With its windows, pull-
/ down menus, buttons, and drag-and-drop operations, OpenWindows provides a consistent, easy-to-use envi-
/ ronment for all types of users. The DeskSet** desktop productivity tools feature integrated, distributed tools
/ from basic time management applications through sophisticated workgroup communications tools. And
/ SunSoft's ToolTalk** interapplication communication solution facilitates information exchange between
/ applications on a single machine, or on multiple machines on a network.

/ The target market for Solaris is the Enterprise 1000. Large companies looking to rightsize their operations
/ will find Solaris an ideal solution for supporting heterogeneous hardware environments ranging from note-
/ books to high-end MP servers using x86, SPARC, and/or PowerPC architectures.

/ G.2 Operating System Version

/ This Appendix describes the first release of Solaris that will support the PowerPC architecture. This version
/ will provide users the same functionality that will be available to Solaris users on the SPARC and x86 archi-
/ tectures. The version number for this release of Solaris has not yet been announced.

/ G.3 Operating System Environment

/ Solaris runs on this PowerPC platform in Little-Endian mode. Solaris supports the ELF object module
/ format as defined in the SVR4 PowerPC ABI. Emulation programs which are not part of the operating
/ system are expected to be available to run applications from other environments, such as Windows, DOS, or
/ Macintosh. Solaris can import the non-native file systems High Sierra (i.e. ISO 9660) FAT (DOS), and S5
/ (UNIX System V).

G.4 Operating System Configuration

Solaris is offered in the following three configurations:

Desktop This configuration is targeted for end-user and developer desktops, for use as clients on a network or as stand-alone workstations. It incorporates the full array of Solaris functionality, and comes with a one- to two-user license.

Workgroup Server This configuration is targeted at departmental servers, for use as print, file, database, or application servers on a small network of up to 100 desktops. The license allows unlimited use but is restricted to platforms with a maximum of (currently) 2 CPUs.

Enterprise Server This configuration supports high-end multiprocessor servers. There is no limit to the number of processors in the server, but the configuration is targeted primarily at MPs with no more than 20 processors. It supplements the basic Solaris functionality with additional components for disk management (RAID), networked backup, advanced network system administration, etc.

G.5 Hardware Configuration Requirements

This section defines the minimum and recommended hardware configurations. In many cases, performance improvements can be realized by configuring systems with larger or faster components. Because these configuration adjustments are application load dependent, no attempt has been made to recommend operationally tuned configurations.

The subsections below describe the configuration recommendations, which are summarized in Table 34.

G.5.1 Processor Subsystem

A PowerPC Reference Platform-compliant system must have a PowerPC compliant processor. Solaris will support PowerPC 601, 603, 604, or compatible PowerPC processors running at supported frequencies. Performance will be better with higher frequencies and with PowerPC processors having performance-enhancing features such as a larger internal cache.

G.5.2 Memory Subsystem

System Memory for a Solaris desktop must be at least 16 MB. Servers may require additional memory. Performance improvements can be realized with additional system memory.

System ROM will not vary between configurations. The minimum size is determined by what a vendor needs to boot the machine into the state expected by the operating system as defined in Section 5.0, “Boot Process and Firmware.”

A cache external to the processor is not required by Solaris, but is recommended for developer and server configurations.

Solaris uses the Non-volatile Memory for configuration and global environment storage. This information is shared between multiple operating systems or versions on the system. Non-volatile Memory requirements are the same for all configurations.

G.5.3 Storage Subsystems

There is no minimum hardfile requirement for Solaris, as diskless operation is fully supported. Alternative configurations provide dataless and cache-only support, requiring small local hardfiles (60 MB). For full installations, an end-user client requires 200 MB of hardfile space, and a developer workstation requires 320 MB of hardfile space. The minimum size for a departmental server is 320 MB. A server may require more space, depending on its function. A file server, for example, normally requires much more hardfile space, in the range of 2 to 4 GB.

A single 1.44-MB 3.5-inch floppy drive is required for all desktops. Solaris also supports a 2.88-MB 2.5-inch floppy drive.

A CD-ROM is required for all Solaris machines. The Solaris distribution media will be CD-ROM only. An acceptable alternative to internal CD-ROM support is an external CD-ROM, or network access to a CD-ROM coupled with booting of desktops over the network. All desktops benefit from high-access-speed CD-ROMs.

G.5.4 Human Interface Subsystem

All configurations should have an alphanumeric input and display device. If a server does not also function as a desktop, it may use a simple device such as a terminal connected to a serial port.

All desktops must have a graphics system capable of showing 640x480x8, although a recommended minimum is 1024x768x8. Higher resolutions and additional color depth are recommended for high-end graphics applications.

A keyboard and pointing device are required for all desktops.

A business audio device is required as part of the standard hardware configuration described in Section 2.0, "Hardware Configuration," but is not required by Solaris for desktops. A higher-function audio device may be substituted.

G.5.5 Connectivity Subsystem

Solaris does not require a serial port, but a serial port is required as part of the standard hardware configuration described in Section 2.0, "Hardware Configuration." Solaris supports EIA/TIA-232-E-compliant serial ports. Any configuration may contain more than one serial port.

Solaris does not require a parallel port. Solaris supports 8-bit bidirectional Compatibility Mode-compliant parallel ports.

No network connection is required, but if present, it is supported by all configurations. Network connectivity is strongly recommended, and is anticipated in the target markets for Solaris.

G.5.6 Expansion Bus Interfaces

Solaris supports the PCI, PCMCIA, and ISA buses. None of these are required, although a PCI bus is strongly recommended.

Solaris supports multiple instances of SCSI interfaces.

Solaris supports IDE access to disks.

G.6 Hardware Configuration Recommendations

Solaris will support additional equipment on PowerPC Reference Platform-compliant expansion buses. In some cases, drivers will have to be supplied by either the system vendor or the device vendor; in others, SunSoft will supply the drivers.

Alternative graphics adaptors may be supplied on PowerPC Reference Platform-compliant machines. Solaris initially supports S3 and Weitek adaptors, although this list will be expanded by SunSoft. Vendors are also encouraged to supply device support for other adaptors.

G.7 Boot Time Abstraction Requirements

Solaris uses boot devices defined by the firmware. Only those devices (video, keyboard, disk) known to the firmware participate in the boot process.

G.8 Hardware Abstraction Layer

Solaris has been ported to both Big-Endian (SPARC) and Little-Endian (Intel x86) processors. The same source code and hardware abstractions used for these processor architectures serves as the base for the PowerPC implementation of Solaris. Within each processor architecture, including PowerPC, Solaris supports multiple platforms.

Loadable modules provide dynamic tailorability for the Solaris kernel. Device drivers, bus interface modules (bus nexus drivers), file system implementations, scheduling algorithms, and STREAMS handling are all contained in Platform-Independent Modules (PIMs), which provide common support for all platforms on a processor architecture that implements a particular feature. For example, a loadable module might support a particular device or controller, a particular bus (e.g. PCI), a particular file system format (e.g. the ISO 9660 format for CD-ROMs), a particular scheduling class (e.g. real time), or a particular networking protocol. Such loadable modules are usually generated from source code that is common across processor architectures, though they might actually be used on only one processor architecture or one platform.

Platform-Specific Modules (PSMs), on the other hand, support functions whose implementation differs from one platform to another. The Kernel Binary Interface (KBI) spells out the interface to which independent hardware providers code platform support modules. The KBI is an extension and formalization of the technology that has been successfully employed for multiprocessor platform support on both SPARC and x86. Typically, supporting a new multiprocessor platform based on an already-supported processor takes less than a month of programmer effort.

A generic distribution of Solaris from SunSoft supports one or more base system configurations. It contains a kernel, a set of device and bus nexus drivers, a complete UNIX System V Release 4 (SVID-compliant) environment, Solaris Deskset tools, system administration software, and subroutine libraries.

The Solaris Device Driver Interface (DDI) provides a well-documented and stable base for independent device driver development. The DDI consists of a common base interface with minor extensions for each of the various processor architectures. Most drivers written for devices that work under Solaris on x86 can be recompiled without source change and run on PowerPC systems that have hardware (bus) support for the same devices. The DDI is supported by a Driver Development Kit (DDK), which consists of descriptions and technical documentation of the interfaces as well as sample drivers.

The generic Solaris distribution will support specified processors in the PowerPC family (support for the 601, 603, and 604 is planned for the initial release), so virtual memory management and TLB and internal cache

/ support are in PIMs provided by SunSoft; coherency of external system memory caches in PowerPC Refer-
/ ence Platform systems is maintained by hardware. Thus, although the KBI does support abstraction of
/ these functions, platform suppliers will usually need to supply a PSM that deals only with differences
/ between their platforms and those supported by the generic distribution in the following areas:

/ a) Interrupt controllers
/ b) Clock and timer functionality
/ c) Power management
/ d) (For multiprocessors only) initialization and inter-processor interrupts

/ Table 33 shows the Solaris components that implement each of the PowerPC Reference Platform
/ abstractions called for in Section 4.0, “Machine Abstractions.”

/ Table 33 (Page 1 of 2). Solaris Abstraction Components

/ **Required Abstraction**	**Solaris Implementation**			
/ /	**Open Firmware**	**Solaris KBI PIM**	**Solaris KBI PSM**	**Solaris DDI**
/ Boot-Time	X			
/ Run-Time Platform Data	X			
/ System Information	X	X		
/ System Memory	X	X		
/ I/O Device Information	X			X
/ Processor Initialization	X		X	
/ I/O Buffer Flushing				X
/ Virtual Memory		X		
/ TLB Flush		X		
/ TLB Reload		X		
/ Cache Management	X	X		
/ Interrupt Handling			X	X
/ DMA			X	X
/ Calendar & Timer Services			X	
/ I/O Addresses	X			X
/ Power Management	X		X	
/ Hardware Fault		X	X	X
/ Bus Nexus Drivers			X	X
/ PCI System Bus			X	
/ ISA System Bus			X	
/ SCSI Framework		X		
/ SCSI Host Bus Adapter				X
/ Device Drivers				X
/ Graphics				X
/ Keyboard				X

Table 33 (Page 2 of 2). Solaris Abstraction Components				
Required Abstraction	**Solaris Implementation**			
	Open Firmware	**Solaris KBI PIM**	**Solaris KBI PSM**	**Solaris DDI**
Super I/O				X
Others				X

G.9 Device Driver Model

A PowerPC Device Driver Development Kit (DDK) available from SunSoft will contain all the documentation and sample code needed to develop device drivers.

G.10 Multiprocessing Model

Solaris is designed to support symmetric multiprocessing. Solaris has no fixed limit to the number of processors, but it is tuned for up to 20. Platform-specific modules may need to be written to support a particular multiprocessing platform. Specifications for writing such modules will be available from SunSoft.

G.11 Application Model

Solaris supports native applications conforming to the System V Interface Definition (SVID), the generic System V application binary interface (gABI), and the PowerPC processor supplement to the System V ABI (PowerPC psABI).

At a source level, the APIs provided by Solaris are identical across all architectures supported by Solaris (SPARC, x86, PowerPC). Applications written to these APIs are independent of the underlying hardware, and require only recompilation to generate binaries that will run on Solaris across all supported architectures.

At a binary level, applications built to run on Solaris for PowerPC systems will run unchanged across all PowerPC platforms supported by Solaris.

Solaris will provide emulation software for running applications native to other operating environments, such as existing DOS and Windows applications built for the x86 architecture.

G.12 Configuration Summary Table

Table 34 gives the minimum configuration information for three Solaris system configurations. This table specifies levels of hardware necessary to run these configurations. In many cases upgrade options are possible. Some of those options and configuration alternatives were described in Section G.5, “Hardware Configuration Requirements.”

Table 34. Hardware Requirements for Operating System Configurations

System Component	Client	Developer	Server
Processor	PowerPC 601, 603, 604, or compatible processor		
System Memory	16 MB	16 MB	16 MB
System ROM	Standard	Standard	Standard
Non-volatile RAM	Standard 4 KB	Standard 4 KB	Standard 4 KB
L2 Cache Memory	Optional	Optional	Optional
Hardfile*	200 MB	320 MB	>=320 MB
Floppy (1.44 MB)	Required	Required	Required
CD-ROM*	Required	Required	Required
Keyboard	Required	Required	Required
Pointing Device	Required	Required	Optional
Audio	Standard	Standard	Standard
Graphics	640x480	1024x768	Optional
Real-Time Clock	Standard	Standard	Standard
Serial Ports	Standard	Standard	Standard
EIA/TIA-232-E	Solaris supports this serial interface		
EIA-422-A	Solaris does not support this serial interface		
Parallel Ports	Optional	Optional	Optional
Compatibility Mode	Solaris supports this parallel interface		
Extended Capabilities Port	Solaris does not support this parallel interface		
Network Adaptors	Optional	Optional	Required
SCSI	Optional	Optional	Optional
IDE	Optional	Optional	Optional
PCI Bus	Optional	Optional	Optional
ISA Bus	Optional	Optional	Optional
PCMCIA	Optional	Optional	Optional
Tape Drive	Optional	Optional	Optional

Legend:

Entry	Definition
Standard	The hardware configuration in Section 2.0, “Hardware Configuration,” requires this component.
Required	This component must be present in systems on which Solaris will run.
Optional	This component is supported by Solaris but is not required.
*	Capability can be provided via a network.

Appendix H. Taligent

Appendix I. PowerPC Supplement to IEEE 1275

I.1 Overview

This appendix specifies the application of *IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware, Core Practices and Requirements* to computer systems that use the PowerPC Instruction Set Architecture, including instruction set-specific requirements and practices for debugging, client program interface and data formats. An implementation of PowerPC Open Firmware must implement the core requirements as defined in [1] and the PowerPC specific extensions described in this binding.

This appendix refers to the publications listed in in Section I.2.1, "References," by number. For example, [2] refers to *The PowerPC Architecture*.

While this appendix addresses the official PowerPC architecture [2], the name "PowerPC" implies compliance only to Book I. The descriptions that follow, and the relevant sections describing translation features for this binding, assume that the system's PowerPC processor(s) implement the entire set of Books I-III. Some PowerPC processors may implement different Book II-III features; such processors may need a variant of this binding describing the differences to the mapping functions, etc.

This appendix defines the binding to PowerPC processors 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 /root must be 1.

Note: Sixty-four-bit processors can follow the specifications contained herein as long as all addresses relevant to Open Firmware are kept within the first 4 GB; i.e. the upper 32 bits of 64-bit addresses are assumed to contain zeros.

I.1.1 Endianess Support

The implementation of PowerPC Open Firmware must support both Big- and Little-Endian system implementations. This section describes features added to the core Open Firmware features to support Bi-Endian booting.

I.1.2 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 Endianess of client programs (e.g. 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 I.8.1.1, "ELF Note Section." When Open Firmware is started, Open Firmware must use the value of **little-endian?** to establish the Endian mode of the 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 than 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 must 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?** must 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.

I.1.3 PowerPC Address Translation Model

This section describes the model that is used for coexistence of Open Firmware and client programs (i.e. operating systems) with respect to address translation.

The following overview of translation is provided so that the issues relevant to PowerPC Open Firmware can be discussed. Details that are not relevant to Open Firmware issues (e.g. protection) are not described in detail; *The PowerPC Architecture* [2], particularly Book III, should be consulted for the details. For the scope of this section, terms will be used as defined in [2], and the 32-bit processor model is assumed.

I.1.3.1 Translation Requirements

The default access mode of storage for loads and stores (i.e. with translation disabled -- referred to as Real-Mode) for PowerPC microprocessors assumes that caches are enabled (in copy-back mode). In order to perform access to I/O device registers, the access mode must be set to Cache-Inhibited, Guarded by establishing a translation with this mode and enabling translation. Thus, even though most of a client program and/or Open Firmware can run with translation disabled, it must be enabled when performing I/O.

I.1.3.2 HTAB Translation

An effective address (EA) of a PowerPC processor is 32 bits wide. Each EA is then translated into a 52-bit virtual address (VA) by prepending a 24-bit virtual segment ID (VSID) to the 28 LSbs of the effective address; the VSID is obtained by indexing into a set of 16 segment registers (SRs) using the 4 MSbs of the EA. Finally, the virtual address is translated into a real address (RA). This is done by mapping the virtual page-number (VPN) (bits 0-39 of the VA) into a real page number (RPN) and concatenating this RPN with the byte offset (bits 40-51 of the VA). The mapping of VPN to RPN involves using a hashing algorithm within the hashed page table (HTAB) to locate a page table entry (PTE) that matches the VPN and using that entry's RPN component. If a valid entry is not found, a data storage interrupt (DSI) or instruction storage interrupt (ISI) is signalled, depending upon the source of the access.

Note: The translation from EA to VA via segment registers means that the granularity of unique effective (Open Firmware virtual) addresses (i.e. for a particular VSID) is 256 MB. This defines the size of, and rules for, the Open Firmware virtual address space described later (see Section I.1.5, "Open Firmware's Virtual-Mode Rules").

This process is not performed for every translation. Processors will typically have a translation lookaside buffer (TLB) that caches the most recent translations, thus exploiting the natural spatial locality of programs to reduce the overhead of address translation. On most PowerPC processors, the TLB updates are performed in hardware. However, the architecture allows an implementation to use a software-assisted mechanism to perform the TLB updates. Such schemes must not affect the architected state of the processor unless the translation fails, i.e. the HTAB does not contain a valid PTE for the VA and a DSI/ISI is signalled.

Note: One unusual feature of this translation mechanism is that valid translations might not be found in the HTAB; the HTAB might be too small to contain all of the currently valid translations. This introduces a level of complexity in the use of address translation by Open Firmware, as discussed below.

I.1.3.3 Block Address Translation

To further reduce the translation overhead for contiguous regions of virtual and real address spaces (e.g. a frame buffer, or all of real memory), the block address translation (BAT) mechanism is also supported by the PowerPC architecture. The block address translation involves the use of BAT registers that contain a block effective page index (BEPI), a block length (BL) specifier and a block real page number (BPRN); the architecture defines four BAT registers for data (DBAT registers) and four BAT registers for instruction (IBAT registers) [4].

Block address translation is done by matching some number of upper bits of the EA (specified by the BL value) against each of the BAT registers. If a match is found, the corresponding BPRN bits replace the matched bits in the EA to produce the RA.

Block address translation takes precedence over HTAB translation; i.e. if a mapping for a storage location is present in both a BAT register and the HTAB, the block address translation will be used.

I.1.4 Open Firmware's Use of Memory

Open Firmware must use the memory resources within the space indicated by the **"my-base"** and **"my-size"** Configuration Variables defined for PowerPC Open Firmware. As described in Section I.8.1.1, "ELF Note Section," a mechanism is defined to enable Open Firmware to determine if its current configuration is consistent with the requirements of the client. If it is not, Open Firmware must set these Configuration Variables appropriately and restart.

A PowerPC Open Firmware binding must support two different addressing models, depending upon the setting of the **real-mode?** Configuration Variable. This variable indicates the Open Firmware addressing mode that a client program expects; **false** (0) indicates Virtual-Mode, **true** (-1) indicates Real-Mode. The default value of **real-mode?** is implementation dependent.

The management of **real-mode?** is analogous to **little-endian?**. Open Firmware determines its addressing mode using the value of **real-mode?**. If the current state of **real-mode?** (and hence, the current state of Open Firmware) is incorrect, it must set **real-mode?** appropriately and reset itself, possibly by executing **reset-all**.

In the following two sections, some conventions in Real-Mode and Virtual-Address translations are described. Subsequent sections describe the assumptions that Open Firmware makes about the state and control of the system in regard to Open Firmware's use of system resources for three Open Firmware interfaces (e.g. Device, User, and Client interfaces).

I.1.4.1 Real-Mode

When **real-mode?** is **true**, Open Firmware must configure its address translation to run in Real-Mode. In Real-Mode, the uses of address translations by Open Firmware and its clients are considered independent; they do not share any translations. All interfaces between the two must pass addresses with the RA of the data. Any data structure shared by Open Firmware and its clients that refer to *virt* addresses in [1], or this binding, must be RAs (i.e. hardware addresses).

Note: In particular, the address of the client interface handler that is passed to the client has to be an RA.

---

[4] The 601 has a single set of BAT registers that are shared by both instruction and data accesses.

The Configuration Variables **"my-base"** and **"my-size"** must indicate the physical memory base and size in Real-Mode. The address base described with **"my-base"** and **"my-size"** does not include address space for I/O devices.

I.1.4.2 Virtual-Mode

When **real-mode?** is **false**, Open Firmware must configure itself to run in Virtual-Mode. In Virtual-Mode, Open Firmware and its client will share a single virtual address space. This binding provides interfaces to allow Open Firmware and its client to ensure that this single virtual address model can be maintained.

The Configuration Variables **"my-base"** and **"my-size"** indicate the virtual address space base and size in Virtual-Mode.

I.1.4.3 Device Interface (Real-Mode)

While Open Firmware is performing system initialization and probing functions, it establishes and maintains its own translations. In particular, it maintains its own HTAB (and/or BAT registers) and handles any DSI/ISI interrupts itself.

Note: In Real-Mode, all translations will be *virt=real*; the primary reason for translation is to allow appropriate I/O accesses.

I.1.4.4 Device Interface (Virtual-Mode)

Open Firmware will establish its own translation environment, handling DSI/ISI interrupts as in the Real-Mode case. However, this environment will, in general, contain *virt≠ real* translations. The virtual address space used by Open Firmware must be compatible with its client.

Note: Since these virtual addresses will be used by the client and/or user interfaces (e.g. for pointers to its code, device tree, etc.), their translations must be preserved until the target OS decides that it no longer requires the services of Open Firmware.

I.1.4.5 Client Interface (Real-Mode)

In Real-Mode, addresses of client data are real; the client must ensure that all data areas referred to across the client interface are valid real addresses. This may require moving data to meet any requirements for contiguous storage areas (e.g. for **read/write** calls). Translation must be disabled before the client interface call is made.

Open Firmware will typically have to maintain its translations in order to perform I/O. Since the client may be running with translation enabled (except for the client interface call), Open Firmware must save the state of all relevant translation resources (e.g. SDR1, BAT registers) and restore them before returning to the client. Likewise, it may take over the DSI/ISI interrupts for its own use (e.g. for doing "lazy" allocation of BAT registers); it must preserve the state of any interrupt vectors for its client.

Since the state of the address translation system is not predictable to any interrupts, the client must ensure that interrupts are disabled before calling the client interface handler.

Once a client program starts execution, physical memory must be managed by the client program. Since Open Firmware does not know which physical memory the client program is using, Open Firmware must request physical memory, if needed, from the the client program (using **alloc-real-mem**).

I.1.4.6 Client Interface (Virtual-Mode)

Client interface calls are essentially "subroutine" calls to Open Firmware. Hence, the client interface executes in the environment of its client, including any translations that the OS has established; e.g. addresses passed in to the client interface are assumed to be valid virtual addresses within the scope of the OS. Any DSI/ISI interrupts are either invalid addresses or caused by HTAB "spills." In either case, the OS has the responsibility for the handling of such exceptions.

Note: Addresses that the Open Firmware internals use will be those that were established by the device interface (or, by subsequent actions of the client or user interface). Thus, the client must preserve these Open Firmware translations if it takes over the virtual memory management function.

In addition to using existing translations, the client interface might require the establishment of new translations (e.g. due to **map-in** calls during **open** time), or the removal of old translations (e.g. during **map-out** calls during **close** time). Since this requires altering the client's translation resources (e.g. HTAB), which might require handling spill conditions, Open Firmware cannot know how to perform these updates to the HTAB.

Hence, there must be *callback* services provided by the client for use by Open Firmware for such actions; see Section I.8.6.2, "Virtual Address Translation Assist Callbacks."

I.1.4.7 User Interface (Real-Mode)

In Real-Mode, Open Firmware has control of the system. As with the client interface in Real-Mode, it must save the state of the translation resources (including interrupt vectors) upon entry and must restore them upon exit.

I.1.4.8 User Interface (Virtual-Mode)

When the user interface is invoked, Open Firmware is responsible for managing the machine. Therefore, it will take over control of any relevant interrupt vectors for its own handling. In particular, it will take over DSI/ISI handling in order to report errors to the user for bad addresses, protection violations, etc. However, as described above, one source of DSI/ISI may simply be HTAB spills. As with the case of **map-in** and **map-out** calls, the user interface cannot know how to handle such spill conditions itself, or even if this is, in fact, a spill versus a bad address.

Hence, there must be a *callback* service provided by the client for use by Open Firmware to resolve such translations; see Section I.8.6.2, "Virtual Address Translation Assist Callbacks."

I.1.5 Open Firmware's Virtual-Mode Rules

In order to let clients (i.e. target operating systems) know where Open Firmware lives in the address space, the following rules must be followed by a PowerPC Open Firmware implementation and by client programs that make use of its client and/or user interfaces.

Open Firmware:

- must maintain its *"translations" "mmu"*-node property (see Section I.5.4, ""mmu" Node Properties")
- must not use BAT registers during client or user interface operations

Client programs:

- must provide callbacks to assist Open Firmware in address translation

I.2 References and Terms

I.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] *The PowerPC Architecture*, published by IBM (ISBN 1-55860-316-6).

[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 the *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] *ANSI/NISO/ISO 9660, Information processing -- Volume and file structure of CD-ROM for information interchange*, published by the 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] *ISO 6429, Information processing -- ISO 7-bit and 8-bit coded character sets -- Additional control functions for character-imaging devices*, published by the International Organization for Standardization.

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

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

I.2.2 Terms

This standard uses technical terms as they are defined in the documents cited in Section I.2.1, "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 PowerPC Open Firmware.

FDISK — Refers to the boot record and partition table format used by MS-DOS, as defined in [4].

linkage area — An area within the stack that is reserved for saving certain registers across procedure calls in PowerPC run-time models. This area is reserved by the caller and is allocated above the current stack pointer (%r1).

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.

procedure descriptor	A data structure used by some PowerPC run-time models to represent a C "pointer to procedure." The first word of this structure contains the actual address of the procedure.
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.
Table of Contents (TOC)	A data structure used by some PowerPC run-time models that is used for access to global variables and for inter-module linkage. When a TOC is used, %r2 contains its base address.
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.

I.3 Data Formats and Representations

The cell size must be 32 bits. Number ranges for *n, u,* and other cell-sized items are consistent with 32-bit, two's-complement number representation.

The required alignment for items accessed with *a-addr* addresses must be four-byte aligned (i.e. a multiple of 4).

Each operation involving a *qaddr* address must be performed with a single 32-bit access to the addressed location; similarly, each *waddr* access must be performed with a single 16-bit access. This implies four-byte alignment for *qaddrs* and two-byte alignment for *waddrs*.

I.4 Packages

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

I.4.1 "Disk-Label" Support Package

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

The process of loading and executing a client program occurs 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 partition information.

I.4.1.1 Partition Formats

PowerPC Open Firmware must support FDISK-type partitions and ISO 9660, FAT-12, and FAT-16 file systems, as shown in the following algorithm:

```
Algorithm for locating boot file
    read sector 0 (bootsector)
    if last 2 bytes of sector are 0AA55h (little-endian)
         if bsMedia == 0F8h  \ FDISK partition on hard drive
             if an explicit partition has been requested
                  select partition number n
             else
                  select bootable partition (80h in peBootable field)
             use directory of the selected partition to locate file
         else (non-partitioned)
             use FAT-12/FAT-16 directory to locate file
    else
    read sector 16.
    if a valid ISO-9660 directory is found
         locate the file, using the ISO-9660 directory.
    else
         FAIL, in an implementation-specific manner.
```

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.

Even though the above algorithm is not dependent on a certain device type, the following formats are strongly recommended for greater portability.

Floppy Disk

It is strongly recommended that floppy disks (1.44/2.88 MB, MFM) be in FAT-12 format, as described in [4].

Hard Disk

It is strongly recommended that hard disks have an FDISK partition map, as described in [4].

Note: Since the boot program is contained in the boot sector 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 that a hard disk and a partition map should be present in the boot sector; any other value indicates a floppy disk.

CD-ROM

It is strongly recommended that CD-ROMs be formatted according to ANSI/NISO/ISO 9660, as described in [7].

I.4.1.2 Program Image Formats

Open Firmware must recognize client programs that are formatted as ELF [3][8] and PE [5]. PE format support is provided only for booting Windows NT; all other clients must 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 algorithm 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 Note Section for the PowerPC Reference Platform
        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.
```

I.4.2 "obp-tftp" Support Package

The *"obp-tftp"* Support Package of an Open Firmware implementation (used to **load** from *"network"* devices) must use the BOOTP protocol, as described in [6].

I.5 Properties

This section describes the standard properties of Open Firmware for PowerPC Open Firmware implementations.

I.5.1 Root Node Properties

The following properties of the root node ("/") must be created by an Open Firmware implementation. Note that 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 must be 1.

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

I.5.2 CPU Node Properties

For each CPU in the system, a CPU node must be defined as a child of "/" (the root). The following properties apply to each of these nodes.

"device_type" Open Firmware standard property. The value of this property for CPU nodes must be *"cpu"*.

"cpu-version" Standard property, encoded as with **encode-int**, that represents the processor type. This value is obtained by reading the Processor Version Register of the CPU.

"clock-frequency" Standard property, encoded as with **encode-int**, that represents the internal processor speed (in hertz) of this node.

"timebase-frequency" Standard property, encoded as with **encode-int**, that represents the rate (in hertz) at which the PowerPC TimeBase and Decrementer registers increment.

I.5.2.1 TLB Properties

Since the PowerPC architecture defines the MMU as being part of the processor, the properties defined by Section 3.6.5 of [1] and the following MMU-related properties must be present under "cpu" nodes.

"tlb-size" Standard property, encoded as with **encode-int**, that represents the total number of TLB entries.

"tlb-sets" Standard property, encoded as with **encode-int**, that represents the number of associativity sets of the TLB. A value of 1 indicates that the TLB is fully associative.

I.5.2.2 Internal (L1) Cache Properties

The PowerPC architecture defines a Harvard-style cache architecture; however, unified caches are an implementation option. All of the PowerPC cache instructions act upon a cache "block" (also referred to as a cache "line"). The internal (also referred to as "L1") caches of PowerPC processors are represented in the Open Firmware device tree by the following properties contained within *"cpu"* nodes.

"cache-unified" This property, if present, indicates that the internal cache has a unified organization. Absence of this property indicates that the internal caches are implemented as separate instruction and data caches.

"i-cache-size" Standard property, encoded as with **encode-int**, that represents the total size (in bytes) of the internal instruction cache.

"i-cache-sets" Standard property, encoded as with **encode-int**, that represents the number of associativity sets of the internal instruction cache. A value of 1 signifies that the instruction cache is fully associative.

"i-cache-block-size" Standard property, encoded as with **encode-int**, that represents the internal instruction cache's block size, in bytes.

"d-cache-size" Standard property, encoded as with **encode-int**, that represents the total size (in bytes) of the internal data cache.

"d-cache-sets" Standard property, encoded as with **encode-int**, that represents the number of associativity sets of the internal data cache. A value of 1 signifies that the data cache is fully associative.

"d-cache-block-size" Standard property, encoded as with **encode-int**, that represents the internal (L1) data cache's block size, in bytes.

“l2-cache”	Standard property, encoded as with **encode-int**, that represents another level of cache in the memory hierarchy. Absence of this property indicates that no further levels of cache are present. If present, its value is the *phandle* of the device node that represents the L2 cache.

I.5.3 External (L2, L3...) Cache Properties

Some systems might include external (L2) cache(s). As with the internal caches, they can be implemented as either Harvard-style or unified. Unlike the L1 properties that are contained within the *“cpu”* nodes, L2 caches are contained within other device tree nodes.

The following properties define the characteristics of such L2 caches. These properties must be contained as a child node of one of the CPU nodes; this is to allow path-name access to the node. All *“cpu”* nodes that share the same L2 cache (including the CPU node under which the L2 cache node is contained) must contain an *“l2-cache”* property whose value is the *phandle* of that L2 cache node.

Note: It is possible to extend this scheme to arbitrary levels of secondary, tertiary, etc. caches. The *“l2-cache”* property must be used in one level of the cache hierarchy to represent the next level. The device node for a subsequent level must appear as a child of one of the caches in the hierarchy to allow path-name traversal.

“device-type”	Open Firmware Standard property; the device-type of L2-cache nodes must be *“cache”*.
“cache-unified”	This property, if present, indicates that the L2 cache has a unified organization. Absence of this property indicates that the L2 caches are implemented as separate instruction and data caches.
“i-cache-size”	Standard property, encoded as with **encode-int**, that represents the total size (in bytes) of the L2 instruction cache.
“i-cache-sets”	Standard property, encoded as with **encode-int**, that represents number of associativity sets of the L2 instruction cache. A value of 1 signifies that the instruction cache is fully associative.
“d-cache-size”	Standard property, encoded as with **encode-int**, that represents the total size (in bytes) of the L2 data cache.
“d-cache-sets”	Standard property, encoded as with **encode-int**, that represents number of associativity sets of the L2 data cache. A value of 1 signifies that the data cache is fully associative.
“l2-cache”	Standard property, encoded as with **encode-int**, that represents another level of cache in the memory hierarchy. Absence of this property indicates that no further levels of cache are present. If present, its value is the *phandle* of the device node that represents the cache at the next level.

I.5.4 "mmu" Node Properties

To aid a client in “taking over” the translation mechanism and still interact with Open Firmware (via the client interface), the client needs to know what translations have been established by Open Firmware. A standard property for “mmu” nodes (i.e. nodes whose *“device_type”* = *“mmu”*) is defined by the PowerPC binding.

"translations" This property, consisting of sets of translations, defines the currently active translations that have been established by Open Firmware (e.g. using *map*). Each set has the following format:

```
( virt size phys mode )
```

Each value is encoded as with **encode-int**.

I.5.5 "/openprom" Node Property

Open Firmware must implement the "halt-address" property under the "/openprom" node. The property value describes the location of physical memory where the firmware saves the halt procedure image. The halt procedure is the self-contained firmware callback service for system power-off or reboot. The halt procedure is discussed further in Section I.8.7, "Halt Callback."

"halt-address" This property describes the physical memory location of the halt procedure. The value has the following format:

```
:prop-encoded-array: address, length
```

If the halt procedure is located within read-write memory, *length* is the number of bytes of memory, encoded as with **encode-int**, occupied by the halt procedure. If the halt procedure is located within read-only memory, *length* is 0.

Note: "Read-only memory" refers to memory that cannot be written in normal operation; ROM or Flash ROM is considered to be "read-only," while RAM, whether or not it is write-protected via an MMU, is considered to be "read-write."

The read-write memory, if any, that the halt procedure occupies must not be included within the "available" property of the "/memory" node.

I.6 Methods

The MMU method defined by Section 3.6.5 of [1] must be implemented by CPU nodes. The value of the **mode** parameter for the relevant methods (e.g. **map**) must be the value that is contained within PTEs that control Write-Through, Cache-Inhibit, Memory-Coherent, Guarded and the 2 protection bits; thus, its format is: WIMGxPP, where x is a reserved bit that must be 0. In order for I/O accesses to be properly performed in a PowerPC system, address ranges that are mapped by **map-in** must be marked as Cache-Inhibited, Guarded.

I.7 Client Interface Requirements

A PowerPC Open Firmware implementation must implement a client interface (as defined in Chapter 6 of [1]) according to the specifications contained within this section.

I.7.1 Calling Conventions

Table 35 (Page 1 of 2). Register Usage Conventions

Register(s)	Value	Notes
%msr	preserved by client interface	
%cr	preserved by client interface	1
%r1-%r2	preserved by client interface	

Table 35 (Page 2 of 2). Register Usage Conventions

Register(s)	Value	Notes
%r3	argument array address on client interface entry	2
	result value (**true** or **false**) on client interface return	2
%r13-%r31	preserved by client interface	
%sprg0-%sprg3	preserved by client interface	
%fpscr	preserved by client interface	
%f0-%f31	preserved by client interface	
%lr, %ctr, %xer	undefined	
Other SPRs	preserved by client interface	3

Notes:

1. Only the non-volatile fields (cr2-cr4) need to be preserved.

2. As defined by Section 6.3.1 of [1]

3 Other special purpose registers

To invoke a client interface service, a client program constructs a client interface argument array as specified in the core Open Firmware document, places its address in %r3 and transfers to the client interface handler, with the return address in %lr. (A typical way of accomplishing this is to copy the client interface handler's address into %ctr and execute a *bctrl*.)

The client interface handler must perform the service specified by the contents of the argument array that begins at the address in %r3, place the return value (indicating success or failure of the attempt to invoke the client interface service) back into %r3, and return to the client program. This is typically done by a Branch to Link Register (*blr*).

The client interface handler must preserve the contents of the Stack Pointer (%r1), TOC Pointer (%r2), Condition Register (%cr), all non-volatile registers (%r13-%r31), and all special purpose registers except %lr, %ctr and %xer.

The preservation of %r2 allows TOC-based client programs to function correctly. Open Firmware must not depend upon whether its client is TOC-based or not. If the client interface handler itself is TOC-based, it must provide for the appropriate initialization of its %r2.

I.8 Client Program Requirements

I.8.1 Client Program Format

The data format of a client program compliant with this specification must be either ELF (Executable and Linkage Format) as defined by [3][8], and extended by Section I.8.1.1, "ELF Note Section," or PE (Portable Executable) as defined by [7]. 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 deals only 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 must be statically linked, requiring no relocation of the image. The program's entry point (e_entry) must contain the address of the first PowerPC instruction of the client program. If the client program uses a TOC, the client program must establish the appropriate value of %r2.

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

I.8.1.1 ELF Note Section

Part of the process of loading a client program involves verifying its environmental requirements (e.g. Endianess and address translation mode) against the current firmware configuration. The client's Endianess 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:

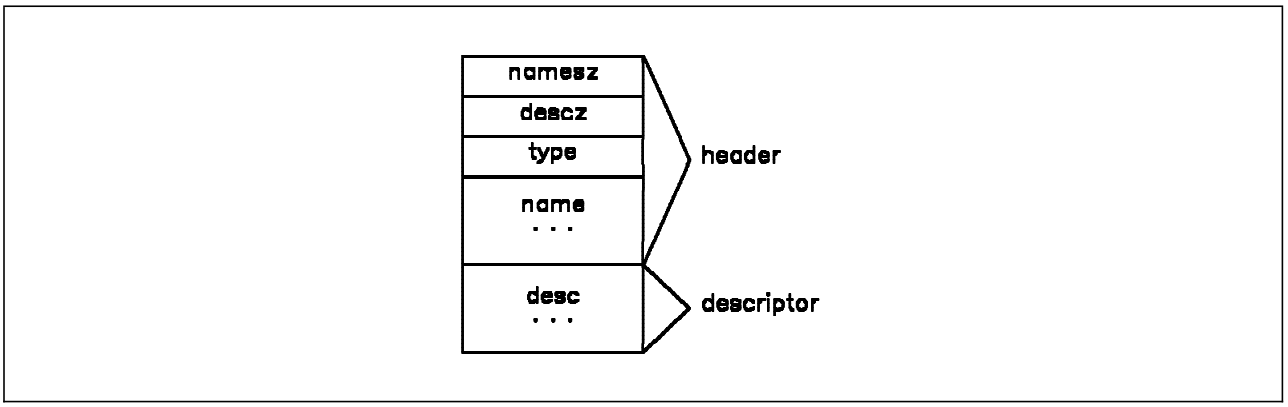

Figure 62. ELF Note Section

Note: The Endian format of the values corresponds to the Endianess 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 PowerPC Reference Platform 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 must be *"PowerPC"*; the *ns.type* field must 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 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 PowerPC Reference Platform Open Firmware
/L  field         ns.real-mode
/L  field         ns.my-base
/L  field         ns.my-size
```

I.8.2 Load Address

The load address is specified by the value of the **load-base** Configuration Variable. At least 4 MB of memory must be available at that address. It is strongly recommended that as much memory as is practical for the particular system be available there, thus allowing the loading of large client programs. 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.

I.8.3 Initial Program State

This section defines the "initial program state," the execution environment that exists when the first machine instruction of a client program of the format specified above begins execution. Many aspects of the initial program state are established by **init-program**, which sets the saved program state so that subsequent execution of **go** will begin execution of the client program with the specified environment.

I.8.3.1 Initial Register Values

Upon entry to the client program, the following registers must contain the following values:

Table 36 (Page 1 of 2). Initial Register Values

Register(s)	Value	Notes
%msr	EE=0, interrupts disabled	1
	PR=0, supervisor state	
	FP=1, floating-point enabled	
	ME=1, machine checks enabled	
	FE0, FE1=0, floating-point exceptions disabled	
	IP see Section I.8.5, "Interrupts"	
	IR, DR see Section I.1.4.1, "Real-Mode"	
	SF=0, 32-bit mode	
	ILE, LE see Section I.1.2, "Bi-Endian Booting"	2
%r1	See Section I.8.3.2, "Initial Stack"	
%r2	0	3
%r3	Reserved	4
%r4	Reserved	4
%r5	See Section I.8.3.3, "Client Interface Handler Address"	
%r6, %r7	See Section I.8.3.4, "Client Program Arguments"	

Table 36 (Page 2 of 2). Initial Register Values		
Register(s)	**Value**	**Notes**
Others	0	

Notes:

1. Open Firmware will typically require the use of external interrupts for its user interface. However, when a client program is invoked, external interrupts must be disabled. If a client program causes the invocation of the user interface, external interrupts may be re-enabled.
2. The 601 processor uses a different mechanism for controlling the Endian mode of the processor. On the 601, the LE bit is contained in the HID0 register; this bit controls the Endian mode of both program and privileged states.
3. Open Firmware does not make any assumptions about whether a client program is TOC-based or not. It is the responsibility of the client program to set %r2 to its TOC, if necessary.
4. As defined in Section 5.6, "Residual Data."

I.8.3.2 Initial Stack

Client programs must be invoked with a valid stack pointer (%r1) with at least 32 KB of memory available for stack growth. The stack pointer must be 16-byte aligned, reserving sufficient room for a linkage area (32 bytes above the address in %r1). If the system is executing in Real-Mode, the value in %r1 is a real (hardware) address; if in Virtual-Mode, the address in %r1 is a mapped virtual address.

I.8.3.3 Client Interface Handler Address

When client programs are invoked, %r5 must contain the address of the entry point of the client interface handler. If the system is executing in Real-Mode, the value in %r5 is a real (hardware) address; if in Virtual-Mode, the address in %r5 is a mapped virtual address.

Note: This address points to the first instruction of the client interface handler, not to a procedure descriptor.

I.8.3.4 Client Program Arguments

The calling program may pass to the client an array of bytes of arbitrary content; if this array is present, its address and length must be passed in registers %r6 and %r7, respectively. For programs booted directly by Open Firmware, the length of this array is zero. Secondary boot programs may use this argument array to pass information to the programs that they boot.

Note: The Open Firmware standard makes no provision for specifying such an array or its contents. Therefore, in the absence of implementation-dependent extensions, such an array will not be passed to a client program executed directly from an Open Firmware implementation. However, intermediate boot programs that simulate or propagate the Open Firmware client interface to the programs that they load can provide such an array for their clients.

Note: **boot** command line arguments, typically consisting of the name of a file to be loaded by a secondary boot program followed by flags selecting various secondary boot and operating system options, are provided to client programs via the *"bootargs"* and *"bootpath"* properties of the *"/chosen*" node.

I.8.4 Caching

The caches of the processor must be enabled when the client program is called. Memory areas allocated on behalf of the client program must be marked as cacheable. Accesses to I/O devices (especially to devices across bus bridges) must be made with the register access words (e.g. *rl@*) using virtual addresses returned by **map-in**.

I.8.5 Interrupts

Open Firmware requires that interrupts be "vectored" to its handlers when it is in control of the processor; this will occur when the user interface is running. Client interface calls are considered to execute in the environment of the client, and hence do not assume ownership of interrupts.

In order for Open Firmware to process interrupts in an efficient manner, an area of memory for the exclusive use by Open Firmware must be reserved by the client program at (real) memory locations 0x1E0...0x1FF.

Open Firmware must save and restore the first location of each interrupt that it wants to "take over"; i.e. whenever Open Firmware needs the use of an interrupt, it must save the current contents of the corresponding entry point and replace that location with a branch to its entry point. When Open Firmware returns control, it must restore the RAM location to its original contents.

I.8.6 Client Callbacks

This section defines the callback mechanism that allows Open Firmware to access services exported to it by the client program. As described in Section 6.3.2.6 (and the glossary entries for *callback* and *$callback*) in [1], the callback mechanism follows the same rules as those of client interface calls; i.e. an argument array is constructed by Open Firmware and the address of that array is passed (via %r3) to the client's callback routine. The address of the callback routine is supplied to Open Firmware by means of the *set-callback* client call.

If the system is running in Real-Mode, the address of the client callback routine must be a real (hardware) address; if it is running in Virtual-Mode, the client callback routine address must be a mapped virtual address.

I.8.6.1 Real-Mode Physical Memory Management Assist Callback

Once the control of physical memory is transferred to the client program, Open Firmware which is running in Real-Mode must use the callback service provided by the client program to allocate physical memory. Client programs that expect Open Firmware to operate in Real-Mode must implement the following physical memory management routine for Open Firmware:

```
alloc_real_mem
     IN: [address] min_addr, [address] max_addr, size, mode
     OUT: error, [address] real_addr
```

This routine allocates a contiguous physical memory of *size* bytes within the address range between *min_addr* and *max_addr*. The *mode* parameter contains the WIMGxPP bits as defined in Section I.6, "Methods." A non-zero error code must be returned if the mapping cannot be performed. If error code is zero (i.e. allocation has succeeded) the routine returns the base address of the physical memory allocated for Open Firmware.

I.8.6.2 Virtual Address Translation Assist Callbacks

As mentioned in Section I.1.5, "Open Firmware's Virtual-Mode Rules," when Open Firmware is running in Virtual-Mode, client programs that take over control of the system's memory management must provide a set of callbacks that implement MMU functions. This section defines the input arguments and return values for these callbacks. The notation follows the style used in Chapter 6 of the Open Firmware specification [1].

```
map
     IN:  [address] phys, [address] virt, size, mode
     OUT: error
```

This routine creates system-specific translation information; this will typically include the addition of PTEs to the HTAB. If the mapping is successfully performed, a value of zero must be placed in the *ret1* cell of the argument array; a non-zero error code must be returned (in *ret1*) if the mapping cannot be performed.

```
unmap
     IN:  [address] virt, size
     OUT: none
```

The system removes any data structures (e.g. PTEs) for the virtual address range.

```
translate
     IN:  [address] virt
     OUT: error, [address] real.hi, [address] real.lo, mode
```

The system attempts to compute the real address (*real*) to which the virtual address (*virt*) is mapped. If the translation is successful, a PTE must be placed into the HTAB for this translation, the resulting real address must be returned in *real.hi,real.lo* and *ret1* must be set to **false** (0). If the translation is not successful, *ret1* must be set to a non-zero error code.

This call must be made when Open Firmware handles a DSI/ISI within the user interface. A successful result of the translate call indicates that Open Firmware can complete the interrupted access; a failure indicates that an access was made to an invalid address.

I.8.7 Halt Callback

The halt callback allows a client program to invoke limited firmware services for turning off or rebooting the machine. The halt procedure must not return control to the client program that invoked it. Consequently, the halt procedure is not required to preserve any machine state on behalf of its caller.

PowerPC Open Firmware must implement the "halt-address" property within the "/openprom" node to support halt callback. Detailed discussion about this property is presented in Section I.5.5, ""/openprom" Node Property."

I.8.7.1 Halt Procedure Calling Conventions

When control is transferred to the halt procedure, the system must be in the Endian mode that was in effect when the firmware last had control of the system. The client program must establish the Real-Mode address translation for the firmware.

I.8.7.2 Halt Procedure Argument

The client program must set GPR3 to point to the halt argument. The halt argument is the address of a null-terminated text string. In the following discussion, this string is called the "halt string."

If the halt string is "power-off," the firmware must turn off the system power to the extent possible. If the firmware cannot turn off the system power, perhaps due to lack of hardware capability, the result is unde-