Linux on Power Architecture Platform Reference

Advance

Version 1.1
24 March 2016

Printed in the United States of America March 2016

IBM Systems
294 Route 100, Building SOM4
Somers, NY 10589-3216

The IBM home page can be found at **ibm.com®**.

Version 1.1
24 March 2016

Table of Contents

List of Tables

List of Figures

About this Document

The purpose of this document is to define the architecture and minimum system requirements on which all LoPAPR platforms are based. These requirements are intended to be precise enough to assure OS compatibility for several operating system (OS) levels (current and “n-1” versions), broad enough to cover workstations through server platforms in single or multiprocessor configurations, and forward-looking enough to allow evolution.

Within the context of this document the term “this architecture” is used to refer to the requirements contained in this document and “LoPAPR” will be used to denote: (1) the architectural requirements specified by this document, (2) the document itself, and (3) as an adjective to qualify an entity as being related to this architecture.

Within the context of this document, “architecture” is defined as the specification of the interface between the platform and the OS. The term “platform” means both hardware and firmware. Device drivers also use this architecture, but require additional definition of the device[1] interfaces to the hardware and OS interfaces within the software.

To the extent that firmware abstracts the hardware interface, it becomes part of the hardware. The firmware to OS interface is defined in this architecture. Two types of firmware are discussed here. Open Firmware (OF) is the initialization or boot code that controls the platform prior to the transfer of control to the OS. Run-Time Abstraction Services (RTAS) is the run-time firmware which provides abstractions to the executing OS. Interfaces within the software or within the hardware are not defined in this document. Where necessary, reference is made to documents where those definitions can be found.

Document Control

Version

The user of this document is responsible for using the most recent version of this document. Please destroy any previous version of this document in your possession.

Page Numbering and End of Document

The pages in this document are numbered sequentially, starting with page 1 at the title page. The last page in this document should say “End of LoPAPR Document.”

1.A device that attaches to an I/O bus is referred to as I/O Adapter, or IOA. See “Glossary” on page 891 for the definition of an IOA.

Goals of This Specification

The specific goals of this specification are as follows:

- To provide an architecture which can be supported by converged AS/RS hardware designs.
- To create a stable platform architecture to be used by platforms based on processors defined by *Power ISA* [1]. Processor implementations based on this architecture include IBM POWER8™ processors and their successors.
- To create an architecture which will allow platforms to operate with previous versions of the OS ("n-1" capability).
- To leverage existing and future industry-standard buses and interfaces.
- To provide a flexible address map. Another key attribute of this specification is the relocatability of devices and subsystems within the Processor Architecture (PA) address space. Subsystem address information, which defines where I/O Adapters (IOAs) reside, is detected by the OF and passed to the OS in the device tree. This architecture accommodates the use of multiple identical buses and IOAs in the same platform without address conflicts.
- To build upon the OF boot environment defined in *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2].
- To provide an architecture which can evolve as technology changes.
- To minimize the support cost for multiple OS versions through the definition of common platform abstraction techniques. Common and compatible approaches to the abstraction of hardware will reduce the burden on hardware developers who produce differentiated machines.
- To architect a mechanism for error handling, error reporting, and fault isolation. This architecture provides for the implementation of more robust systems, if desired by the system developers.
- To architect a mechanism for Dynamic Reconfiguration of the hardware.
- To provide an architecture which allows for the logical partitioning of system resources, in order to execute multiple concurrent OS instances.

Audience for This Document

This document defines the platform and system requirements for designing LoPAPR platforms. This document is the primary source of information that a platform, OS, or hardware component developer would need to create compatible products. Additional requirements are defined by the industry standards referenced in this document.

This document describes the platform to OS interface which must be provided in these platforms. Platform designers must assemble components and firmware which match this interface. Also, the document defines minimum system configuration requirements. Platform designers must meet or exceed these minimums to build a standard platform.

This document must be used by those designing compatible software including the OS, boot software, or firmware. If a function is supported, software developers must provide support for the interfaces described in this document. This software must provide the mandatory functions and capabilities as described in the requirements in this document. However, this document does not limit this software from going beyond the specification through software tailored to specific hardware. For example, the OS must implement the interface to the required abstracted interfaces to hardware, but the OS may also implement fast paths to specific hardware that it recognizes.

Suggested Reading

The “Bibliography” on page 889 provides a full list of references and ordering information for these references. Within this document, the number of the reference in the bibliography is placed after the citation in brackets, “[nn]”.

This document assumes the reader has an understanding of computer architecture in general, the Processor Architecture, the various Peripheral Component Interconnect (PCI) specifications, and OF. Some understanding of the current personal computer and workstation architectures is also useful. A list of suggested background reading includes:

- *Power ISA* [1]. Note that this specification is referred to as the “Processor Architecture,” or “PA,” in the body of this document.
- *PCI Local Bus Specification* [18]
- *PCI-X Protocol Addendum to the PCI Local Bus Specification* [21]
- *PCI Express Base Specification* [22]
- *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2] and relevant bindings

Conventions Used in This Document

Requirement Enumeration

Within the body of this document requirements are clearly defined by separate paragraphs beginning with a bold text sequence number. A list of all these are available as a separate document. These requirements may point to other standards documents, or figures or tables which conveniently show the requirement. The referenced material becomes part of the requirements in this document. Users of this document must comply with these requirements to build a standard platform. Other material in this document is supportive description of these requirements, architecture notes, or implementation notes. Architecture or implementation notes are flagged with a descriptive phrase—for example, “Hardware Implementation Note”— and followed by indented paragraphs. The descriptive material and notes provide no additional requirements and may be used for their information content.

Big-Endian Numbering

Big-Endian numbering of bytes and bits is used in this document, unless indicated otherwise. Numbering of bits starts at 0 for the most significant bit and continues to the least significant bit, unless indicated otherwise. All data structures used for communicating between the OS and the platform (for example, RTAS and hypervisor calls) will be in Big-Endian format, unless otherwise designated.

Hypertext Links

This document makes use of hypertext links. Cross references, Table of Contents entries, List of Tables entries, and List of Figures entries are all clickable hypertext links, to make navigation of the document easier.

Specific Terms

In this document:

- The term “Processor Architecture” (PA) is used to mean compliance with the requirements specified in *Power ISA* [1].
- The term “processor” is used as a general term to mean “processor,” or “processor core,” or “thread of a multi-threaded processor design.” In any case where it makes a difference, the specific terminology for that case is used.
- The term “real” used in relationship with addresses is a generic term that means “processor real address” when the platform is running in SMP (non-LPAR) mode and partition “logical real address” when the platform is running in LPAR mode.
- The term “PCI” is used as a general term to describe the most recent versions of all forms of PCI standards. In cases where there are significant differences between individual PCI standards, the following terminology is used to differentiate between the PCI standards: conventional PCI, PCI-X, and PCI Express.

Typographical Conventions

Typographical conventions used in this document are described in Table 1, “Typographical Conventions,” on page 34.

Table 1. Typographical Conventions

Text Element	Description of Use
Rn-m-x.	A requirement number. The number “n” indicates a requirement sequence number and is changed only when it is necessary change the sequence numbers of one or more requirements on an update. The number “m” is the section that the requirement appears in, and the number “x” is the number of the requirement within that section. The sequence numbers are automatically generated and are restarted at the beginning of each new section. Sequence numbers for existing requirements will not change unless it becomes necessary to insert a requirement between other requirements or to add a section that renumbers other sections (at which point, the number “n” will be incremented for the next publication of the document).
Italics	◆ Used for emphasis such as the first time a new term is used. ◆ Indicates a book title. ◆ Indicates PA instruction mnemonics. ◆ Indicates RTAS function, field names, and parameter names
`Courier Bold`	◆ Indicates OF properties, methods, configuration variables, node names and encode functions (for example, **`encode-int`** and **`encode-string`**). In addition, OF properties are enclosed in quotes. ◆ Indicates NVRAM partition names
`Courier` (not bold, enclosed in quotes)	Indicates a character string value.
0xnnnn	Prefix to denote hexadecimal numbers.
0bnnnn	Prefix to denote binary numbers.
nnnnn	Numbers without a prefix are decimal numbers.
0xF... FFF100	This hexadecimal notation represents a replication of the hexadecimal character to the right of the ellipsis to fill out the field width. For example, the address 0xF... FFF100 would be 0xFFFFF100 on a processor with a 32-bit address bus or 0xFFFFFFFFFFFFF100 on a processor with a 64-bit address bus.
0:9	Ranges of bits are specified by two numbers separated by a colon. The range includes the first number, all numbers in between, and the last number.
0xm-0xn	A range of addresses or values within the document is always inclusive, from m up to and including n.

Table 1. Typographical Conventions ***(Continued)***

Text Element	Description of Use
<token>	This notation means the character or character string named within the less than and greater than symbols is used in the place of the symbols and name. For instance, the property name **"ibm,sensor-**<*token*>**"** indicates the set of properties **"ibm,sensor-9000"**, **"ibm,sensor-9001"**, . . .
Reserved For Compatibility	This notation is a placeholder used to reserve numbering (for example, chapter, section and requirement numbers) so that subsequent numbering remains consistent with document changes.
NULL vs. null	NULL designates an ASCII NULL string (0x00). The term "null" indicates the empty set.

1 Introduction

This architecture specification provides a comprehensive computer system platform-to-software interface definition, combined with minimum system requirements, that enables the development of and software porting to a range of compatible industry-standard computer systems from workstations through servers. These systems are based on the requirements defined in *Power ISA* [1][1]. The definition supports the development of both uniprocessor and multiprocessor system implementations.

A key attribute and benefit of this architecture is the ability of platform developers to have degrees of freedom of implementation below the level of architected interfaces and therefore have the opportunity for adding unique value. This flexibility is achieved through architecture facilities including: (1) device drivers; (2) Open Firmware (OF); (3) Run-Time Abstraction Services (RTAS); and (4) hardware abstraction layers. The role of items 1 and 4 are described in separate operating system (OS) documentation. The role that items 2 and 3 play in this architecture will be described in subsequent paragraphs and chapters.

This architecture combines leading-edge technologies to create a superior computing platform. By design, it supports a wide range of computing needs including personal productivity, engineering design, data management, information analysis, education, desktop publishing, multimedia, entertainment, and database, file, and application servers. This architecture effectively leverages industry-standard I/O through the PCI bus. Systems based on this architecture are expected to offer price/performance advantages and to address the expected growth in computing performance and functionality.

Architecture Note: In modern platforms, designers may choose between various PCI topology standards. This architecture uses the term "PCI" as a general term to describe the most recent versions of all forms of PCI standards including any approved Engineering Change Requests (ECRs) against them. In cases where there are significant differences between individual PCI standards, the following terminology is used to differentiate between the PCI standards.

- The term "conventional PCI" refers to behavior or features that conform to the most recent version of the *PCI Local Bus Specification* [18], including any approved ECRs against it.
- The term "PCI-X" refers to behavior or features that conform to the most recent version of the *PCI-X Protocol Addendum to the PCI Local Bus Specification* [21], including any approved ECRs against it.
- The term "PCI Express" refers to behavior or features that conform to the most recent version of the *PCI Express Base Specification* [22] including any approved ECRs against it. In addition, the terms "bus," "bridge" and "PCI Host Bridge (PHB)" used in relation to "PCI" throughout this architecture may refer to a PCI Express "link," "switch," and "root complex" respectively.

1.The term "Processor Architecture" (PA) is used throughout this document to mean compliance with the requirements specified in *Power ISA* [1].

1.1 Platform Topology

To the experienced computer designer and system manufacturer, much of the content of this architecture will be familiar. A typical desktop topology is shown in Figure 1, “Typical Desktop Topology,” on page 39. This topology consists of a single PA processor, volatile System Memory, and a single Host Bridge providing a PCI Bus. The PCI Bus provides for connection of I/O adapters (IOAs). See “Glossary” on page 891 for the definition of an IOA.

A more complex general topology is shown in Figure 2, “General Platform Topology,” on page 40. All platforms consist of one or more PA processors, a volatile System Memory separate from other subsystems, and a number of IOAs, which may initiate transactions to System Memory. The processors are linked over the primary processor bus/switch to each other, to the System Memory, and to one or more Host Bridges. In general, IOAs do not connect to the primary processor bus/switch. The Host Bridges connect to secondary buses which have IOAs connected to them. In turn, one or more bus bridges may be employed to tertiary buses (and beyond) with additional IOAs connected to them. Typically, the bus speeds and throughput decrease and the number of supportable loads increases as one progresses from the primary processor bus to more remote buses.

There are variations on these topologies, which are likely to occur and are therefore worth describing below. This architecture describes interfaces, not implementation. The logical software model must remain the same, even if the physical topology is different.

- In a smaller platform, the Host Bridge and/or the memory and/or an IOA may be integrated into a single chip. In this case, the topology would not look like Figure 1, “Typical Desktop Topology,” on page 39 from a chip point of view, but instead would be integrated onto the single chip.
- In a larger platform, secondary buses may be implemented, with two or more Host Bridges, as two or more parallel expansion buses for performance reasons. Similarly, tertiary buses may be two or more parallel expansion buses off each secondary bus. This is indicated by the ellipses near the Host Bridge and the Bus Bridge.
- In a high performance platform, with multiple processors and multiple memories, a switch may be employed to allow multiple parallel accesses by the processors to memory. The path through the switches would be decided by the addressing of the memory.

Each of the following chapters provides information necessary to successfully implement compliant systems. It is recommended that the reader become thoroughly familiar with the contents of these chapters and their references prior to beginning system or software development. It is anticipated that standard chip sets will simplify the development of compliant implementations consistent with the topologies shown below and will be available from third-party industry sources.

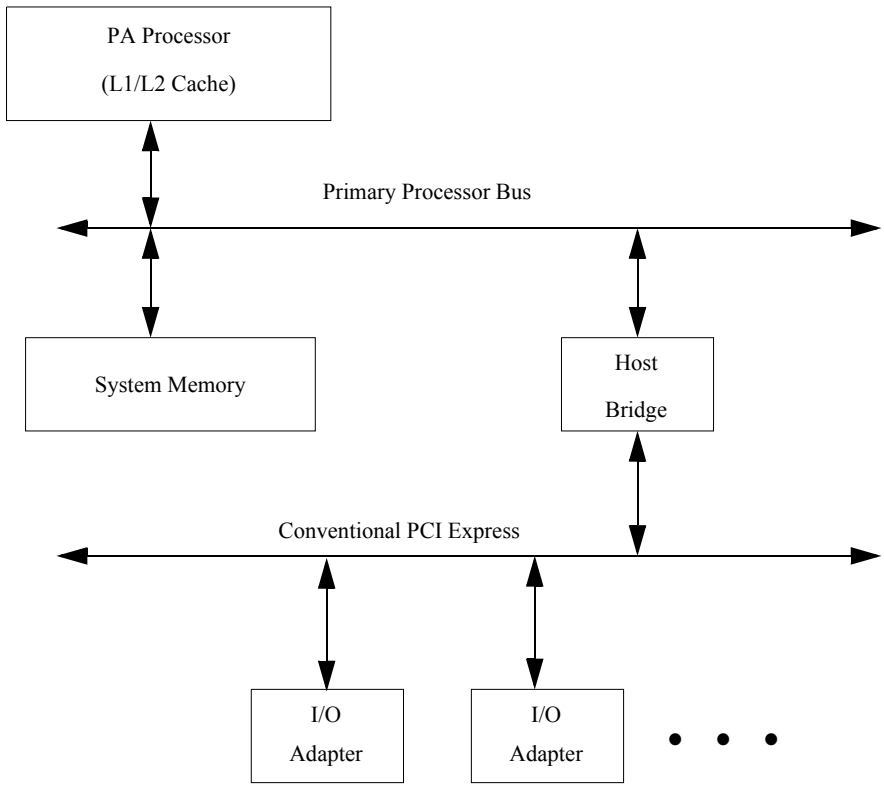
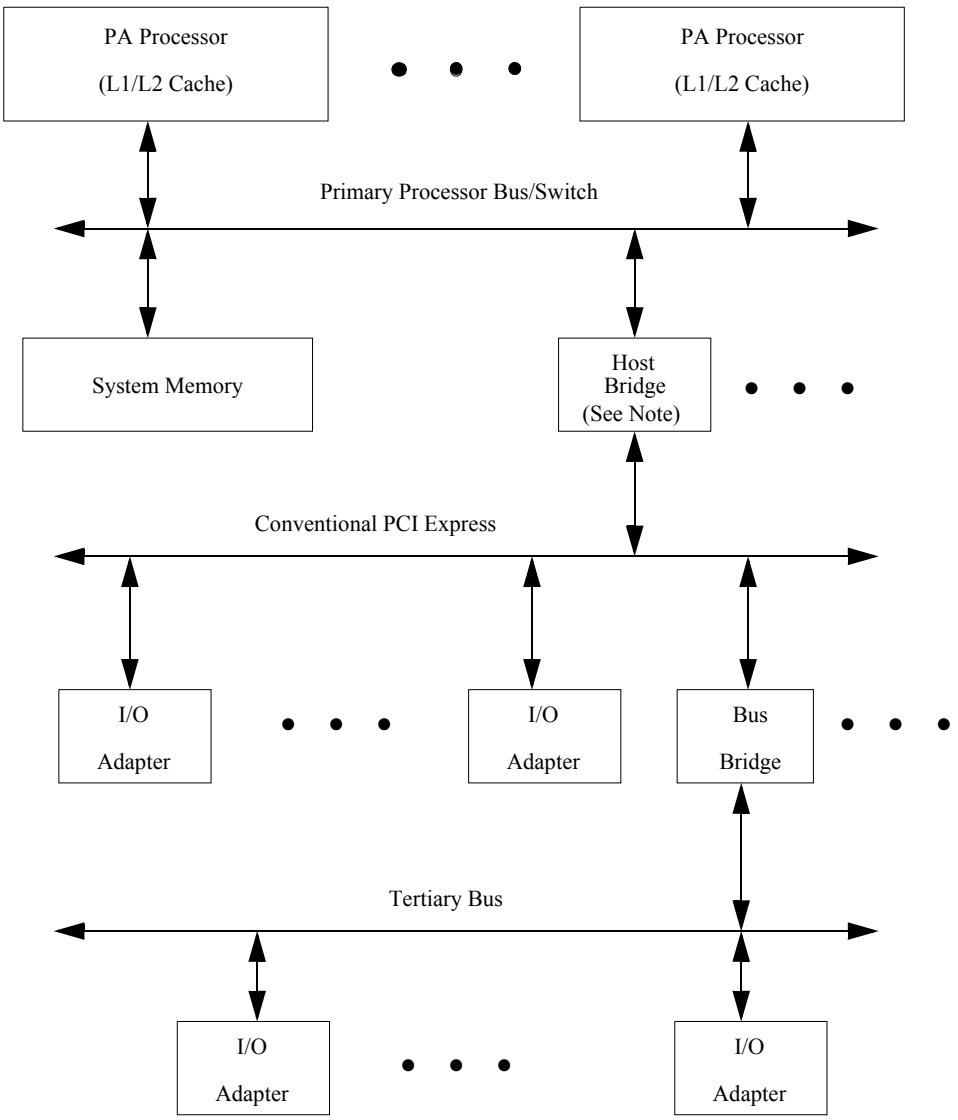


Figure 1. Typical Desktop Topology



Note: To enable the implementation of a large number of I/O adapters in a large system, the Host Bridge may be split into two pieces -- a Hub and a Bridge -- with the two connected by a cable, thus enabling the I/O adapters to be housed at a distance from the main processor enclosure.

Figure 2. General Platform Topology

2 System Requirements

This chapter gives an operational overview of LoPAPR systems and introduces platform specific software and/or firmware components that are required for OS support. This chapter also addresses some system level requirements that are broad in nature and are fundamental to the architecture described in later chapters. Lastly, a table of requirements is presented as a guide for platform providers.

2.1 System Operation

2.1.1 Control Flow

Figure 3, “Phases of Operation (example),” on page 41 is an example of typical phases of operation from power-on to full system operation to termination. This section gives an overview of the processes involved in moving through these phases of operation. This section will introduce concepts and terms that will be explained in more detail in the following chapters. Most requirements relating to these processes will also appear in later chapters.

The discussion in this chapter will be restricted to systems with a single processor. Refer to Chapter 11, “The Symmetric Multiprocessor Option,” on page 321 for the unique requirements relating to multiprocessor systems.

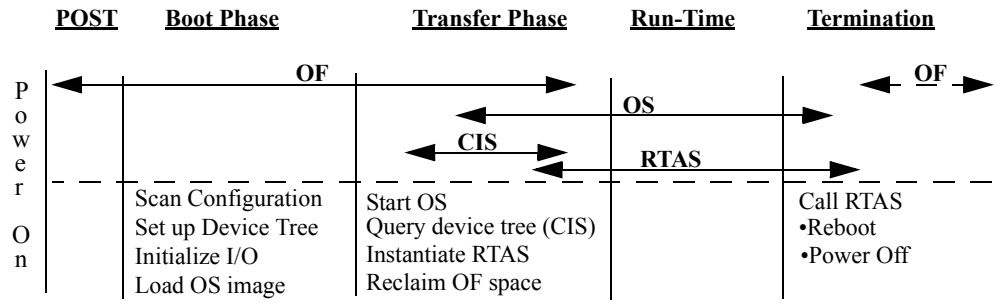


Figure 3. Phases of Operation (example)

2.1.2 POST

Power On Self Test (POST) is the process by which the firmware tests those areas of the hardware that are critical to its ability to carry out the boot process. It is not intended to be all-inclusive or to be sophisticated in how it relates to the user. Diagnostics with these characteristics will generally be provided as a service aid.

Platform Implementation Note: The platform may choose to utilize a service processor to assist in the implementation of functions during various phases of operation. The service (or support) processor is not a requirement of this architecture, but is usually seen in the larger systems.

2.1.3 Boot Phase

The following sections describe the boot phase of operation. The fundamental parts of the boot phase are:

1. Identify and configure system components.
2. Generate a device tree.
3. Initialize/reset system components.
4. Locate an OS boot image.
5. Load the boot image into memory.

2.1.3.1 Identify and Configure System Components

Firmware is generally written with a hardware in mind, so some components and their configuration data can be hard-coded. Examples of these components are: type of processor, cache characteristics, and the use of imbedded components on the planar. This hardcoding is not a requirement, only a practical approach to a part of this task.

R1–2.1.3.1–1. The firmware must, by various means, become aware of all components in the system associated with the boot process and configure or reset those components into a known state (components include, for example, buses, bridges, I/O Adapters (IOAs)[1], and I/O devices).

R1–2.1.3.1–2. The firmware must obtain certain system information which is necessary to build the OF device tree from “walking” the I/O buses (for example, identification of IOAs and bridges).

2.1.3.2 Generate a Device Tree

R1–2.1.3.2–1. The firmware must build a device tree and the OS must gain access to the device tree through Client Interface Services (CIS).

R1–2.1.3.2–2. Configuration information (configuration variables) which are stored in non-volatile memory must be stored under the partition names **`of-config`** or **`common`**, depending on the nature of the information (see Chapter 8, “Non-Volatile Memory,” on page 265).

2.1.3.3 Initialize/Reset System Components

The OS requires devices to be in a known state at the time control is transferred from the firmware. Firmware may gain control with the hardware in various states depending on what has initiated the boot process.

- Normal boot: Initiated by a power-on sequence; all devices and registers begin in a hardware reset state.

1.See “Glossary” on page 891 for the definition of an IOA.

- Reboot: Device state is unpredictable at the start of a reboot.

The hardware reset state for a device is an *inactive* state. An *inactive* state is defined as a state that allows no system level activity; there can be no bus activity, interrupt requests, or DMA requests possible from the IOA that is in a reset state. Since the OS may configure devices in a manner that requires very specific control over these functions to avoid transitory resource conflicts, these functions should be disabled at the *device* and not at a central controlling agent (for example, the interrupt controller). Devices that do not share any resources may have these resources disabled at a system level (for example, keyboard interrupts may be disabled at the interrupt controller in standard configurations).

R1–2.1.3.3–1. IOAs must adhere to the reset states given in Table 2, “IOA Reset States,” on page 43 when control of the system is passed from firmware to an OS.

Table 2. IOA Reset States

Bus	IOAs Left Open by OF	Other IOAs
PCI	Interrupts not active No outstanding I/O operations IOA is configured	The IOA is *inactive*: •I/O access response disabled •Memory access response disabled •PCI master access disabled •Interrupts not active IOA is reset (see note)
System	Configured per OF device tree •Interrupts inactive •DMA inactive •No outstanding I/O operations	The IOA is in hardware reset state (see note) or *inactive*: •Interrupts inactive •DMA inactive

R1–2.1.3.3–2. The platform must include the root node OF device tree property **`“ibm,pci-full-cfg”`** with a value of 1 and configure the configuration registers of all PCI IOAs and bridges as specified by Requirement R1–9.1.8–1.

R1–2.1.3.3–3. Prior to passing control to the OS, the platform must initialize all processor registers to a value which, if accessed, will not yield a machine check.

R1–2.1.3.3–4. Prior to passing control to the OS, the platform must initialize all registers not visible to the OS to a state that is consistent with the system view represented by the OF device tree.

R1–2.1.3.3–5. During boot or reboot operations and prior to passing control to the OS, the platform must initialize the interrupt controller.

R1–2.1.3.3–6. Hardware must provide a mechanism, callable by software, to hard reset all processors and I/O subsystems in order to facilitate the implementation of the RTAS *system-reboot* function.

Platform Implementation Note: The platform is required to reset the interrupt controller to avoid inconsistency among the states of IOAs, the interrupt controller, and software interrupt handler routines. The reset state is shown in Table 2, “IOA Reset States,” on page 43.

Software and Firmware Implementation Note: The conventional PCI configuration registers are further described in the *PCI Local Bus Specification* [18] and are copied into OF properties described in the *PCI Bus binding to: IEEE Std 1275-1994, Standard for Boot (Initialization, Configuration) Firmware* [6]. PCI-X configuration registers are further described in the *PCI-X Protocol Addendum to the PCI Local Bus Specification* [21]. PCI Express configuration registers are further described in the *PCI Express Base Specification* [22]. PCI-X IOAs and bridges

and PCI Express IOAs, bridges, and switches are treated the same as conventional PCI IOAs and bridges for purposes of generation of OF properties.

Software and Firmware Implementation Note: In reference to Requirement R1–2.1.3.3–3, generally the initial value of processor registers is contained in the processor binding. However, some processors have deviations on register usage. Also, since some register implementation is optional, all processors are not the same.

2.1.3.4 Locate an OS Boot Image

The OS boot image is located as described in Appendix B, “LoPAPR Binding,” on page 661. A device and filename can be specified directly from the command interpreter (the **`boot`** command) or OF will locate the image through an automatic boot process controlled by configuration variables. Once a boot image is located, the device path is set in the device tree as the **`“bootpath”`** property of the **`chosen`** node.

The devices searched by the automatic boot process are those contained in the **`boot-device`** configuration variable. Implementations may choose to limit the number of boot device entries that are searched. The root node device tree property **`“ibm,max-boot-devices”`** communicates the number of **`boot-device`** entries that the platform processes.

If multi-boot (multiple bootable OSs residing on the same platform) is supported, a configuration variable instructs the firmware to display a multi-boot menu from which the OS and bootpath are selected. See Appendix B, “LoPAPR Binding,” on page 661 for information relating to the multiboot process.

R1–2.1.3.4–1. The platform must supply in the OF root node the **`“ibm,max-boot-devices”`** property.

2.1.3.5 Load the Boot Image into Memory

After locating the image, it is loaded into memory at the location given by a configuration variable or as specified by the OS load image format.

2.1.3.6 Boot Process

The boot process is described in Appendix B, “LoPAPR Binding,” on page 661. Steps in the process are reviewed here, but the authoritative and complete description of the process is included in Appendix B, “LoPAPR Binding,” on page 661. Figure 4, “Boot Process,” on page 45 is a depiction of the boot flow showing the action of the f1, f5, and f6 function keys. The figure should only be used as an aid in understanding the requirements for LoPAPR systems.

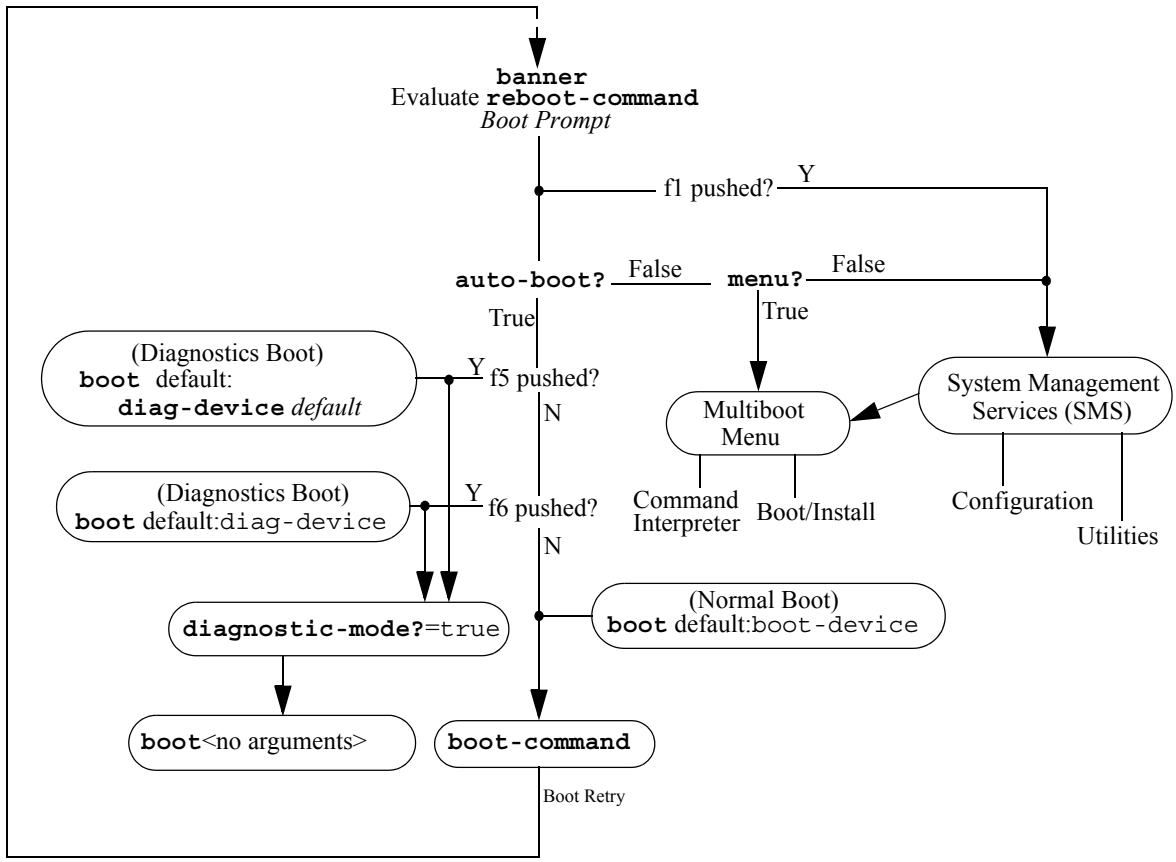


Figure 4. Boot Process

2.1.3.6.1 The Boot Prompt

R1–2.1.3.6.1–1. After the **banner** step of the boot sequence, the platform display must present a clearly visible graphical or text message (*boot prompt*), and must provide a reaction window of at least 3 seconds that prompts the user to activate various options including the f1, f5, and f6 control keys detailed in this document.

R1–2.1.3.6.1–2. The functions provided by f1, f5 and f6 described in this chapter must be equivalently provided by the tty numeral keys 1, 5, and 6, respectively when a serial terminal is attached.

R1–2.1.3.6.1–3. The *boot prompt* must identify the platform and communicate to the user that there are options that may be invoked to alter the boot process.

2.1.3.6.2 The Menus

Once the *boot prompt* is displayed, the System Management Services (SMS) menu can be invoked. SMS provides a user interface for utilities, configuration, and the Multiboot Menu (as introduced in Appendix B, “LoPAPR Binding,” on page 661) for boot/install and the OF command interpreter.

The Multiboot menu is formatted so that block devices that currently contain boot information are most easily selected by the user. Because of the serial nature of byte devices, they should not be opened unless specifically included in a boot list. The user may also wish to add devices to the boot-device and/or diag-device configuration variables (boot lists) that currently do not contain boot information. The Multiboot menu presents these devices in a secondary manner.

If the Multiboot Menu boot/install option is chosen, OF will execute the `bootinfo.txt<boot-script>` of the selected OS, and if the user elects to make this the default, the **`boot-command`** variable will be set equal to the contents of `bootinfo.txt<boot-script>`.

R1–2.1.3.6.2–1. The SMS menu must provide a means to display the Multiboot Menu.

R1–2.1.3.6.2–2. If, after the *boot prompt* is displayed, **`auto-boot?`** = `false` and **`menu?`**=`true`, the firmware must display the Multiboot Menu directly.

R1–2.1.3.6.2–3. The Multiboot Menu must present all potential boot device options, differentiating block devices that contain locatable *bootinfo objects*.

R1–2.1.3.6.2–4. Firmware must evaluate all *bootinfo objects* at each invocation of the Multiboot Menu to ensure that any modifications made by the OS will be included.

R1–2.1.3.6.2–5. The Multiboot Menu must provide a means to enter the currently selected boot option into the desired location within the **`boot-device/boot-file`** or **`diag-device/diag-file`** configuration variables.

R1–2.1.3.6.2–6. The platform must provide a means to delete individual boot options from the **`boot-device/boot-file`** and **`diag-device/diag-file`** configuration variables.

R1–2.1.3.6.2–7. The Multiboot Menu must provide an option for the user to select whether or not to return to the Multiboot Menu on each boot.

Firmware Implementation Note: Returning to the Multiboot Menu on reboot is controlled with the **`auto-boot?`** and **`menu?`** configuration variables.

2.1.3.6.3 The f1 Key

The boot process is further controlled by the **`auto-boot?`** and **`menu?`** OF configuration variables and the f1 key.

R1–2.1.3.6.3–1. If, after the *boot prompt* is displayed, function key f1 is pushed or if **`auto-boot?`** = `false` and **`menu?`**=`false`, the firmware must display the System Management Services (SMS) menu.

R1–2.1.3.6.3–2. The default value for the **`auto-boot?`** configuration variable must be `true`.

R1–2.1.3.6.3–3. The default value for the **`menu?`** configuration variable must be `false`.

2.1.3.6.4 The f5 and f6 Keys

If **`auto-boot?`** = `true`, the commands specified by the **`boot-command`** configuration variable are executed.

If the **`boot`** command has no arguments, IEEE 1275 states that the arguments are determined as follows:

- Normal Boot - If the **`diagnostic-mode?`** FCode function returns `false`, the boot device is given by **`boot-device`** and the default boot arguments are given by **`boot-file.`**
- Diagnostics Boot - If the **`diagnostic-mode?`** FCode function returns `true`, the boot device is given by **`diag-device`** and the default boot arguments are given by **`diag-file`**.

Platform Implementation Note: **`boot-device`**, **`boot-file`**, **`diag-device`** and **`diag-file`** are potentially multi-entry strings. The **`boot-command`** searches the devices specified in **`boot-device/diag-device`** in the order defined by the string for the **`boot-file/diag-file`** to load into system memory. Failure occurs only if no corresponding file is found/usable on any of the specified devices.

Platforms give the user the ability to control the boot process further with function keys f5 and f6 (within the window described in Requirement R1–2.1.3.6.1–1).

R1–2.1.3.6.4–1. If, after the *boot prompt* is displayed, function key f5 is pushed (and **auto-boot?** = true), then **diagnostic-mode?** must return true and the *default diagnostic device* as defined in Requirements R1–2.1.3.6.4–4 and R1–2.1.3.6.4–5 must be used to locate bootable media.

R1–2.1.3.6.4–2. If, after the *boot prompt* is displayed, function key f6 is pushed (and **auto-boot?** = true), then **diagnostic-mode?** must return true and **diag-device** must be used to locate the boot image, else if **diag-device** is empty, then its default as defined in Requirements R1–2.1.3.6.4–4 and R1–2.1.3.6.4–5 must be used to locate bootable media.

R1–2.1.3.6.4–3. **boot-command** must default to **boot**<with no arguments>.

R1–2.1.3.6.4–4. **boot-device** and **diag-device** must default to the first devices of each type that would be encountered by a search of the device tree.

R1–2.1.3.6.4–5. The search order for the **boot-device** and **diag-device** defaults must be floppy, cdrom, tape, disk, network.

R1–2.1.3.6.4–6. **boot-file** must default to <null>.

R1–2.1.3.6.4–7. **diag-file** must default to diag.

Note: Requirement R1–2.1.3.6.4–1 provides a method to invoke stand-alone diagnostics or to start reinstallation without going through the menus. Requirement R1–2.1.3.6.4–2 provides a method to boot with on-line diagnostics.

Software Implementation Note: Pressing either f5 or f6 at the correct time will cause the contents of **diag-file** to be set into the **"bootargs"** property of the **chosen** node of the device tree. The OS can recognize a diagnostics boot request when it finds the "diag" substring in **"bootargs"**.

2.1.3.6.5 CDROM Boot

If the CDROM is the first bootable media found in the devices listed in the bootlist (**boot-device** strings), the CDROM should boot without having to enter optional file specification information or using the f5 function key normally used for diagnostic boot. This is accomplished by having the appropriate bootinfo.txt file specification in the CDROM entry in the bootlist.

R1–2.1.3.6.5–1. CDROM entries for the default OF **boot-device** and **diag-device** configuration variables must include the standard block device bootinfo.txt file specification as documented in Appendix B, "LoPAPR Binding," on page 661 (\ppc\bootinfo.txt).

2.1.3.6.6 Tape Boot

Boot from tape is defined in Appendix B, "LoPAPR Binding," on page 661.

2.1.3.6.7 Network Boot

The user selects from a list of network devices on the Multiboot Menu and then selects the *boot* option. The user may be prompted for network parameters (IP addresses, etc.) which are set as arguments in **boot-device** by the firmware. If the BOOTP protocol is used, the BOOTREPLY packet contains the network parameters to be used for subsequent transmissions (see *Open Firmware Recommended Practice:OBP-TFTP extension* [4] for details of this process).

R1–2.1.3.6.7–1. If network boot is selected, firmware must provide a means for the user to specify or override network parameters.

2.1.3.6.8 Service Processor Boot

In platforms with a service processor, the user may call for a boot using a local/remote connection to the service processor. The particular port used for this remote session is sent to the firmware in a status message after the service pro-

cessor finishes POST. The port is identified in the **"stdin"** and **"stdout"** properties in the **chosen** node of the OF device tree.

2.1.3.6.9 Console Selection

During the boot process, firmware establishes the console to be used for displaying status and menus. The following pseudocode describes the console selection process:

IF there is a configuration change detected for "display", "keyboard" or "mouse" type devices OR if a console selection timeout occurred on the last boot,

  IF Firmware can locate a supported console device,

    Firmware will prompt the user to select a console and wait approximately 60 seconds for a valid response.

    IF a valid response is received within the 60 second timeout,

      Use the selected console.

    ELSE choose the Primary Serial Port.

  ELSE choose the Primary Serial Port.

ELSE

  IF input-device and output-device are valid,

    Firmware will choose these devices for the console.

  ELSE choose the Primary Serial Port.

R1–2.1.3.6.9–1. If a console has been selected during the boot process, firmware must set the **"stdin"** and **"stdout"** properties of the **chosen** node to the ihandles of this console's input and output devices prior to passing control to the OS.

2.1.3.6.10 Boot Retry

For boot failures related to firmware trying to access a boot device, it is appropriate for the platform to retry the boot operation, especially in the case of booting from a network device. However, in platforms which have a service processor, there are several other types of detected errors for which a reboot retry may be appropriate; for example, checkstops or loss of communication between firmware and the service processor. To ensure that the user policy is followed, the coordination and counting of retry attempts need to be interlocked between the service processor and boot firmware. The most straightforward way to implement this is to have the boot firmware inform the service processor of all failed boot attempts, and let the service processor initiate the system reset (as it also would for checkstops or hangs). This way the service processor can easily manage the retry count and initiate a service dial-out if the boot retry limit is exceeded.

R1–2.1.3.6.10–1. Platform Implementation: In platforms with service processors, retry of failed boot operations must be coordinated between boot firmware and the service processor, to ensure correct counting and handling of reboot retries according to the service processor configuration reboot policies.

2.1.3.6.11 Boot Failures

Failure to boot occurs only when no corresponding file is found which is usable on any device specified in the **boot-device**, **boot-file**, **diag-device**, or **diag-file** string being used.

R1–2.1.3.6.11–1. If an error occurs in a boot device preventing boot from that device, and after all defined retries have occurred, the failure must be reported as a POST error.

R1–2.1.3.6.11–2. If a boot device is physically missing or lacks a boot record (for example, if a CDROM is not present in a CDROM drive), then a POST error must be generated for this case, must not result in the calling out of a boot device as being defective, and must not result in a hardware service repair action to the device.

R1–2.1.3.6.11–3. In Requirement R1–2.1.3.6.11–2, if it is not possible for a device to distinguish between an actual device error, as opposed to a missing device or boot record, then a POST error must be generated that indicates the possible causes of the failure to boot from the device, and this POST error must not imply that a hardware service repair action is required for the boot device.

Implementation Note: All device errors of the same type may be consolidated into a single POST log entry with multiple location codes listed if needed. This architecture anticipates remote support center notification of hardware errors. It is the intention that only definitive boot device errors will be reported as requiring hardware repair. This is meant to prevent service calls for systems for non-hardware errors such as no tape in a tape drive.

2.1.3.6.12 Persistent Memory and Memory Preservation Boot (Storage Preservation Option)

Selected regions of storage, or Logical Memory Blocks (LMBs), may be optionally preserved across client program boot cycles. These LMBs are denoted by the presence of the **"ibm,preservable"** property in their OF device tree **/memory** node. The client program registers the LMB with the platform using the *ibm,manage-storage-preservation* RTAS call if it wants the contents of the storage preserved across client boot cycles (see also Section 7.4.4, "Managing Storage Preservation," on page 240). The architectural intent of this facility is to enable client programs to emulate persistent storage. This is done by a client program registering preservable LMBs. Then, after a subsequent boot cycle (perhaps due to error or impending power loss) the presence of the **"ibm,preserved-storage"** property in the **/RTAS** node of the device tree indicates to the client program that it has preserved memory. When the client program detects that it has booted with preserved storage and that it might be necessary to preserve the storage for long term, the client program is responsible for copying the preserved data to long term persistent storage medium, and then clearing the registration of the preserved LMBs to prevent potential corruption of the persistent storage medium due to subsequent failures.

Upon reboot after such an operation, the **"ibm,request-partition-shutdown"** property is provided in the **/rtas** node with a value of 2, indicating that the client program should save appropriate data and shutdown the partition.

Implementation Note: How areas get chosen to be marked as preservable is beyond the scope of this architecture.

2.1.4 Transfer Phase

The image is prepared for execution by checking it against certain configuration variables; this may result in a reboot.

Once the OS gains control, it may use the CIS interface to learn about the platform contents and configuration. The OS will generally build its own version of this configuration data and may discard the OF code and device tree in order to reclaim the space used by OF. A set of platform-specific functions are provided by Run-Time Abstraction Services (RTAS) which is instantiated by the OS invoking the **instantiate-rtas** method of the RTAS OF device tree node.

R1–2.1.4–1. If any device tree property is presented that contains a phandle value to identify a certain node in the device tree, the device tree node so identified must contain the **"ibm,phandle"** property, and the value of the **"ibm,phandle"** property must match the phandle value in the property identifying that node.

R1–2.1.4–2. If the **"ibm,phandle**" property is present in a device tree node, the OS must use this value, and not the phandle value returned by a client interface service, to associate this node with a device tree property that uses a phandle value to identify this node.

R1–2.1.4–3. An OS must not assume that the **"ibm,phandle"** property, if present, corresponds to the phandle used by or returned by OF client interface services. A phandle value passed to a client interface service as an

argument must have been obtained by use of a client interface service, and not from a device tree property value.

Note: If the **"ibm,phandle"** property exists, there are two "phandle" namespaces which must be kept separate. One is that actually used by the OF client interface, the other is properties in the device tree making reference to device tree nodes. These requirements are written to maintain backward compatibility with older FW versions predating these requirements; if the **"ibm,phandle"** property is not present, the OS may assume that any device tree properties which refer to this node will have a phandle value matching that returned by client interface services. It will be necessary to have the OSs ready for this requirement before the firmware implementation.

2.1.5 Run-Time

During run-time, the OS has control of the system and will have RTAS instantiated to provide low-level hardware-specific functions.

2.1.6 Termination

Termination is the phase during which the OS yields control of the system and may return control to the firmware depending on the nature of the terminating condition.

2.1.6.1 Power Off

If the user activates the system power switch, power may be removed from the hardware immediately (switch directly controls the power supply) or software may be given an opportunity to bring the system down in an orderly manner (power management control of the power switch).

If power is removed from the hardware immediately, the OS will lose control of the system in an undetermined state. Any I/O underway will be involuntarily aborted and there is potential for data loss or system damage. A shut-down process prior to power removal is highly recommended.

In most power managed systems, power switch activation is fielded as a power management interrupt and the OS (through RTAS) is able to quiesce the system before removing power. The OS may turn off system power using the RTAS *power-off* function.

2.1.6.2 Reboot

The OS may cause the system to reset and reboot by calling the RTAS *system-reboot* function.

2.2 Firmware

R1–2.2–1. Platforms must implement OF as defined in Appendix B, "LoPAPR Binding," on page 661.

R1–2.2–2. The OF User Interface must include the following methods as specified in *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], Section 7.6: **.registers**, **to**, **load**, **go**, **state-valid** and **init-program**.

R1–2.2–3. Platforms must implement the Run-Time Abstraction Services (RTAS) as described in Chapter 7, "Run-Time Abstraction Services," on page 107.

R1–2.2–4. OSs must use OF and the RTAS functions to be compatible with all platforms.

2.3 OS Installation

Installation of OSs will be accomplished through the Multiboot Menu as follows:

1. The system boots or reboots normally; the user enters the Multiboot Menu by one of the methods described herein.
2. The Multiboot Menu presents a list of all installation devices.
3. The user selects “install” and an installation device from the menu; firmware locates the bootinfo object or install image on the selected installation device.
4. Firmware will execute **`init-program`** and, if a bootinfo object was found, firmware parses it, replaces the <boot-script> *entities* with appropriate values and executes the script.
5. The OS gets control and selects the target device.
6. After the install process is determined to be successful, the OS updates variables such as **`boot-device`**, **`boot-file`**, and **`boot-command`**.
7. The OS adds the **`bootinfo-nnnn`** configuration variable to the NVRAM **`common`** system partition.

R1–2.3–1. The Multiboot Menu must provide an option for OS installation that lists all possible installation devices.

R1–2.3–2. After the install process is determined to be successful, the OS must set **`boot-device`**, **`boot-file`**, and **`boot-command`**.

2.3.1 Tape Install

The OF definition of installation from tape is defined in Appendix B, “LoPAPR Binding,” on page 661.

2.3.2 Network Install

Network install follows the same process as network boot with the exception that after installation is complete, the OS will write **`boot-device`** with the target device information.

R1–2.3.2–1. If network install is selected, firmware must provide a means for the user to override default network parameters.

2.4 Diagnostics

IBM Power® system platforms may use IBM AIX® Kernel-based *stand-alone* diagnostics as their multi-OS common diagnostics package. Since AIX will run on other vendors’ platforms which might not have permission to use AIX diagnostics, the **`“ibm,aix-diagnostics”`** property indicates that AIX diagnostics are permitted (see Section B.6.2.1, “Root Node Properties,” on page 673).

R1–2.4–1. If AIX diagnostics are supported on a platform, then the firmware for that platform must include the property **`“ibm,aix-diagnostics”`** in the **`root`** node.

Software Implementation Note: Each OS may implement an OS-specific *run-time* diagnostics package, but should, for purposes of consistency, adhere to the error log formats in Section 10.3, “RTAS Error and Event Information Reporting,” on page 289.

2.5 Platform Class

The **“ibm,model-class”** OF property is defined to classify platforms for planning, marketing, licensing, and service purposes (see Section B.6.2.1, “Root Node Properties,” on page 673).

R1–2.5–1. The **“ibm,model-class”** property must be included in the platform’s **root** node.

2.6 Security

Platforms will provide the user with options for a *Power On Password* (*POP*) and a *Privileged Access Password* (*PAP*) and will have some optional physical security features.

R1–2.6–1. Platform Implementation: Platforms must provide a *Power On Password* (POP) capability which, when enforced, controls the user’s ability to power-on and execute the configured boot sequence.

R1–2.6–2. Platform Implementation: Platforms must provide a *Privileged Access Password* (PAP) capability which, when enforced, controls the user’s ability to alter the boot sequence using f5/f6, and to enter SMS and the Multiboot Menu.

R1–2.6–3. Platform Implementation: If the PAP is absent or <NULL>, but the POP is non-<NULL>, then the POP must act as the PAP.

R1–2.6–4. Platform Implementation: Platforms must accept the PAP as a valid response to a request to enter the POP.

R1–2.6–5. Platform Implementation: If there is a key switch implemented with a *secure* position, the system must not complete the boot process regardless of the state of POP and PAP when the switch is in this position.

R1–2.6–6. Platform Implementation: If a key switch is implemented and the switch is in the *maintenance (service)* position, the POP and PAP must *not* be enforced.

R1–2.6–7. Platform Implementation: Platforms, except for rack mounted systems, must provide a locking mechanism as an option which prevents the removal of the covers.

R1–2.6–8. Platform Implementation: Platforms, except for rack mounted systems, must provide a tie-down mechanism as an option which prevents the physical removal of the system from the premises.

R1–2.6–9. Platform Implementation: Passwords and keyswitch positions must be implemented in a manner that makes their values accessible to both OF and the service processor.

R1–2.6–10. Platform Implementation: The OF configuration variable `security-password` must be maintained to be equivalent to the Privileged Access Password (PAP).

R1–2.6–11. Platform Implementation: If the PAP and `security-password` are absent or <NULL>, `security-mode` must be set to **“none”**, otherwise `security-mode` must be set to **“command”**.

R1–2.6–12. Platform Implementation: If `security-mode` is set to any value other than **“none”** (such as **“command”** or **“full”**), it must be treated as **security-mode** = **command**.

Platform Implementation Notes:

1. As defined here, the PAP and **security-password** are stronger than as specified in IEEE 1275 for **security-mode** = command in that they are required for *any* command line operations, including **go** and **boot**. The PAP and **security-password** are not required to boot the system with default parameters, however, and in this sense the intent of **security-mode** = command is achieved. There is currently no implementation of **security-mode** = full.

2. If a service processor is provided, the requirements relating to passwords are applicable in the service processor environment. Service processor documentation refers to the POP as the *General User Password* and the PAP as the *Privileged User Password*.

2.7 Endian Support

LoPAPR platforms operate with either Big-Endian (BE) or Little-Endian (LE) addressing. In Big-Endian systems, the address of a word in memory is the address of the most significant byte (the “big” end) of the word. Increasing memory addresses will approach the least significant byte of the word. In Little-Endian (LE) addressing, the address of a word in memory is the address of the least significant byte (the “little” end) of the word.

All data structures used for communicating between the OS and the platform (for example, RTAS and hypervisor calls) are Big-Endian format, unless otherwise designated.

R1–2.7–1. Platforms must by default operate with Big-Endian addressing.

R1–2.7–2. Platforms that operate with Little-Endian addressing must make System memory appear to be in Little-Endian format to all entities in the system that may observe that image, including I/O.

Platform Implementation Notes:

1. Some hardware (for example, bridges, memory controllers, and processors) may have modal bits to allow those components to be used in platforms which operate in Little-Endian mode. In this case, the hardware or firmware will need to set those bits appropriately.

2. Requirement R1–2.7–2 may have an impact on the processor chosen for the platform.

2.8 64-Bit Addressing Support

A *64-bit-addressing-capable platform* is defined as one capable of supporting System Memory and Memory Mapped I/O (MMIO) configured above 4 GB (greater than 32 bits of real addressing). This means that all hardware elements in the topology down to the Host Bridges are capable of dealing with a real address range greater than 32 bits, and all Host Bridges are capable of providing a translation mechanism for translating 32-bit I/O bus DMA addresses. All platforms compliant with LoPAPR version 2.3 and beyond are required to be 64-bit-addressing-capable.

A *64-bit-addressing-aware OS* is an OS that can deal with a real address space larger than 4GB. It must handle the 64-bit processor page table format (required of all OSs), and must understand Host Bridge mechanisms and Host Bridge OF methods for supporting System Memory greater than 4 GB. All OSs compliant with LoPAPR version 2.3 and beyond are required to be 64-bit-addressing-aware.

2.9 Minimum System Requirements

This section summarizes the minimum hardware and functionality required for LoPAPR compliance.

The term *portable* is used in this document to describe that class of systems that is primarily battery powered and is easily carried by its user.

The term *personal* is used in this document to describe that class of systems that is bound to a specific work area due to its size or power source, and whose use is generally restricted to a single direct user or a small set of users.

The term *server* is used in this document to describe that class of systems that supports a multi-user environment, providing a particular service such as file storage, software repository, or remote processing capability.

Each of these classes may have unique requirements due to the way it is used or which OS it generally employs and, for this reason, the requirements in this document my have qualifiers based on the type of system being developed.

R1–2.9–1. *(Requirement moved to Table 4, "IBM Server Required Functions and Features," on page 58)*

R1–2.9–2. A means of attaching a diskette drive must be provided (may be through a connector or over a network) and the drive must have the following characteristics:

a. Media sense: Implementations must allow polling of the drive up to 100x per second to determine the presence of media in the drive.

b. Must accept media of type: 3.5" 1.44 MB MFM

R1–2.9–3. A means of attaching a CD-ROM drive must be provided (may be through a connector or over a network) and the drive must have the following characteristics:

a. ISO9600 compliant

b. Supports multi-session

R1–2.9–4. When a keyboard is provided, it must be capable of generating at least 101 scan codes.

R1–2.9–5. When a mouse is provided, it must have at least two buttons.

R1–2.9–6. The capability to generate a tone must be provided on portable and personal platforms, and on server platforms which are not housed in rack enclosures.

R1–2.9–7. A Real Time Clock (RTC) must be provide which must have the following characteristics:

a. Is non-volatile

b. Runs continuously

c. Has a resolution of at least one second

2.10 Options and Extensions

Options are features that are covered by this architecture, but are not necessarily required to be present on a given platform. Platforms that implement options are required to conform to the definitions in this architecture, so that an aware OS environment can recognize and support them. Some options may be *required* on some platforms. Refer to Table 3, "LoPAPR Optional Features," on page 55 for the disposition of currently defined options, including requirements for implementation of some of these options on some platforms. Note that in this table, "optional" does not mean "not required;" see the description column of the table for more information.

An extension is a feature that is added to this architecture and is required on all platforms developed after a specified effective date.

Options and extensions will normally need to be dormant or invisible in the presence of a non-aware OS environment. In general, this means that they come up passively; that is, they are initialized to an inactive state and activated by an aware OS.

R1–2.10–1. Extensions and options must come up passively unless otherwise specified in this architecture.

R1–2.10–2. Extensions and options that affect the OS interface to the platform must be identified, when present, through some architected means, such as OF device tree properties.

It is the responsibility of the product development teams to keep the “usage” columns of Table 3, “LoPAPR Optional Features,” on page 55 up to date,

Table 3. LoPAPR Optional Features

Option Name	Usage		Description
	Base	IBM Server	
Usage Legend: NS = Not Supported; O = Optional (see also Description); OR = Optional but Recommended; R = Required; SD = See Description			
Symmetrical Multiprocessing (SMP)	O	R	Required on MP platforms.
Multiboot	O	O	Required to support multiple versions of an OS.
PCI Hot Plug DR	O	OR	See Chapter 13, “Dynamic Reconfiguration (DR) Architecture,” on page 355 for more information.
Logical Resource Dynamic Reconfiguration (LRDR)	O	OR	See Section 13.7, “Logical Resource Dynamic Reconfiguration (LRDR),” on page 377.
Enhanced I/O Error Handling (EEH)	OR SD	R	See Section 4.1, “I/O Topologies and Endpoint Partitioning,” on page 71 and Section 4.4.1, “Enhanced I/O Error Handling (EEH) Option,” on page 85. Required for platforms that implement LPAR, regardless of the number of partitions (Requirements R1–14.3–1 and R1–14.3–2).
Error Injection (ERRINJCT)	O	R	Required of servers which implement the EEH option.
Logical Partitioning (LPAR)	O	R	See Chapter 14, “Logical Partitioning Option,” on page 385.
Bridged-I/O EEH Support	O	R	EEH support for I/O structures which contain PCI to PCI bridges or PCI Express switches. See Section 4.4.3, “Bridged-I/O EEH Support Option,” on page 91. Required if EEH is supported.
PowerPC External Interrupt	R SD	R SD	May be virtualized; See Chapter 6, “Interrupt Controller,” on page 101.
EXTI2C	O	O	See Section 7.3.9.2, “Ibm,exti2c,” on page 166. For support of I^2C buses.
Firmware Assisted NMI (FWNMI)	R	R	See Section 7.3.14, “Firmware Assisted Non-Maskable Interrupts Option (FWNMI),” on page 204.
System Parameters	R	R	See Section 7.3.16, “System Parameters Option,” on page 207.
Capacity on Demand (CoD)	O	O	See Section 7.3.16.4, “Capacity on Demand (CoD) Option,” on page 213.
Predictive Failure Sparing	O	O	See Section 7.3.16.4.2, “Predictive Failure Sparing with Free Resources,” on page 214.
Converged Location Codes	R	R	The Converged Location Codes option is required on all platforms being developed. See Section 12.3, “Hardware Location Codes,” on page 327 and Requirement R1–12.3–1.
Shared Processor LPAR (SPLPAR)	O	O	See Section 14.11, “Shared Processor LPAR Option,” on page 446.
Reliable Command/Response Transport	O	O	See Section 17.2.3.1, “Reliable Command/Response Transport Option,” on page 637.
Logical Remote DMA (LRDMA)	O	O	See Section 17.2.3.2, “Logical Remote DMA (LRDMA) Option,” on page 642.
Interpartition Logical LAN (ILLAN)	O	O	See Section 16.4, “Interpartition Logical LAN (ILLAN) Option,” on page 551.
ILLAN Backup Trunk Adapter	O	O	See Section 16.4.6.1, “ILLAN Backup Trunk Adapter Option,” on page 571.

Table 3. LoPAPR Optional Features ***(Continued)***

Option Name	Usage		Description
	Base	IBM Server	
Usage Legend: NS = Not Supported; O = Optional (see also Description); OR = Optional but Recommended; R = Required; SD = See Description			
ILLAN Checksum Offload Support	O	O	See Section 16.4.6.2, “ILLAN Checksum Offload Support Option,” on page 572.
Checksum Offload Padded Packet Support	O	O	See Section 16.4.6.2.3, “Checksum Offload Padded Packet Support Option,” on page 574.
Virtual SCSI (VSCSI)	O	O	See Section 16.5, “Virtual SCSI (VSCSI),” on page 575.
Virtual FC (VFC)	O	O	See Section 16.7, “Virtual Fibre Channel (VFC) using NPIV,” on page 590.
Storage Preservation	O	NS	See Section 2.1.3.6.12, “Persistent Memory and Memory Preservation Boot (Storage Preservation Option),” on page 49 and Section 7.4.4, “Managing Storage Preservation,” on page 240.
Client Vterm	O	R	Required of all platforms that support LPAR, otherwise not implemented. Provides a virtual “Asynchronous” IOA for connecting to a server Vterm IOA, the hypervisor, or HMC (for example, to a virtual console). See Section 16.6, “Virtual Terminal (Vterm),” on page 582 for more information.
Server Vterm	O	O	Allows a partition to serve a partner partition's client Vterm IOA.
NUMA Associativity Information	OR	OR	See Chapter 15, “Non Uniform Memory Access (NUMA) Option,” on page 505.
Performance Tool Support	O	NS	Provides access to platform-level facilities for performance tools running in a partition on an LPAR system. See Section 7.4.5, “ibm,lpar-perftools RTAS Call,” on page 242.
MSI (Message Signaled Interrupt)	SD	SD	Required for all platforms that support PCI Express.
ILLAN Buffer Size Control	O	O	See Section 16.4.6.3, “ILLAN Buffer Size Control Option,” on page 574.
Virtual Management Channel (VMC)	O	O	See Section 17.7, “Virtual Management Channel (VMC),” on page 714.
Partition Suspension	O	O	Requires the Logical Partitioning, LRDR, and Update OF Tree options.
Partition Hibernation	O	O	Allows a partition to sleep for an extended period; during this time the partition state is stored on secondary storage for later restoration. Requires the Partition Suspension, ILLAN, and VASI options.
Partition Migration	O	O	Allows the movement of a logical partition from one platform to another; the source and destination platforms cooperate to minimize the time that the partition is non-responsive. Requires the Partition Suspension, ILLAN, and VASI options.
Thread Join	O	O	Allows the multi-threaded caller to efficiently establish a single threaded processing environment.
Update OF Tree	O	O	Allows the caller to determine which device tree nodes changed due to a massive platform reconfiguration as happens during a partition migration or hibernation.
Virtual Asynchronous Services Interface (VASI)	O	O	Allows an authorized virtual server partition (VSP) to safely access the internal state of a specific partition. See Section 17.8, “Virtual Asynchronous Services Interface (VASI),” on page 715 for more details. Requires the Reliable Command/Response Transport option.
Virtualized Real Mode Area (VRMA)	O	O	Allows the OS to dynamically relocate, expand, and shrink the Real Mode Area.
TC	O	O	Allows the OS to indicate that there is no need to search secondary page table entry groups to determine a page table search has failed. See Section 14.12.2, “Virtualizing the Real Mode Area,” on page 473 for more details.
Configure Platform Assisted Kernel Dump	O	O	Allows the OS to register and unregister kernel dump information with the platform.

Table 3. LoPAPR Optional Features *(Continued)*

Option Name	Usage		Description
	Base	IBM Server	
Usage Legend: NS = Not Supported; O = Optional (see also Description); OR = Optional but Recommended; R = Required; SD = See Description			
I/O Super Page	O	OR	Allows the OS to specify I/O pages that are greater than 4 KB in length.
Subordinate CRQ (Sub-CRQ) Transport	O	O	Support for the Subordinate CRQs as needed by some Virtual IOAs. See Section 17.2.3.3, “Subordinate CRQ Transport Option,” on page 645.
Cooperative Memory Over-commitment (CMO)	O	O	The CMO option allows for partition participation in the over-commitment of logical memory by the platform. See Section 14.12.3, “Cooperative Memory Over-commitment Option (CMO),” on page 474.
Partition Energy Management (PEM)	O	O	Allows the OS to cooperate with platform energy management. See Section 14.14, “Partition Energy Management Option (PEM),” on page 492.
Multi-TCE-Table (MTT)	O	O	Support for the Multi-TCE-Table Option. See Section 14.5.4.2.4, “H_PUT_TCE_INDIRECT,” on page 421.
Virtual Processor Home Node (VPNH)	O	O	Provides substantially consistent virtual processor associativity in a shared processor LPAR environment. See Section 14.11.6, “Virtual Processor Home Node Option (VPHN),” on page 467.
IBM Active Memory™ Compression	O	O	Allows the partition to perform active memory compression.
Virtual Network Interface Controller (VNIC)	O	O	See Section 17.3, “Virtual Network Interface Controller (VNIC),” on page 652.
Expropriation Subvention Notification	**O**	**O**	Allows OS notification of a cooperative memory overcommitment page fault see Section 14.12.3.8, “Expropriation/Subvention Notification Option,” on page 485.
Boost Modes	O	O	Allows the platform to communicate and the availability of performance boost modes along with any ability to manage the same. See Section 7.3.16.20, “Performance Boost Modes Vector,” on page 231
Platform Resource Reassignment Notification (PRRN)	O	O	See Section 15.6, “Platform Resource Reassignment Notification Option (PRRN),” on page 509
Dynamic DMA Windows (DDW)	O	O	Allows the creation of DMA Windows above 4 GB. See Section 7.4.10, “DMA Window Manipulation Calls,” on page 259.
Universally Unique Partition Identification Option (UUID)	O	O	See Section 7.3.16.21, “Universally Unique IDentifier,” on page 233
Platform Facilities Option (PFO)	O	O	See Section 14.5.4.1.1, “H_REMOVE,” on page 408, Section 14.5.4.1.2, “H_ENTER,” on page 410, and Section 14.15, “Platform Facilities,” on page 499
Extended Cooperative Memory Overcommittment (XCMO)	O	O	Introduces additional cooperative memory overcommitment functions see Section 14.12.3, “Cooperative Memory Over-commitment Option (CMO),” on page 474

2.11 IBM LoPAPR Platform Implementation Requirements

The tables in this section detail specific product requirements which are not defined as an “option” in this architecture. The intent is to define base requirements for these products, over and beyond what is specified in Table 3, “LoPAPR Optional Features,” on page 55 and elsewhere in this architecture.

In addition, any options that are unique to specific implementations (that is, not general usage), and which do not appear in Table 3, “LoPAPR Optional Features,” on page 55, are listed in this section.

It is the responsibility of the product development teams to keep these tables up to date.

2.11.1 IBM Server Requirements

This section talks to the requirements for IBM LoPAPR Compliant server platforms.

R1–2.11.1–1. For all IBM LoPAPR Compliant Platforms: The platform must implement the options marked as "required" in the IBM Server column of Table 3, "LoPAPR Optional Features," on page 55 and the additional functions as indicated in *Table 4, "IBM Server Required Functions and Features," on page 58* (that is, the "Base" column of Table 3, "LoPAPR Optional Features," on page 55 is not sufficient).

Table 4. IBM Server Required Functions and Features

Function/Feature	Effective Date	Description
All IOA device drivers EEH enabled or EEH safe	6/2004	Required even for systems running with just one partition.

It is the responsibility of the product development teams to keep *Table 4, "IBM Server Required Functions and Features," on page 58* up to date.

2.12 Behavior for Optional and Reserved Bits and Bytes

Behavior of the OSs and platforms for bits and bytes in this architecture that are marked as reserved or optional are defined here.

R1–2.12–1. Bits and bytes which are marked as "optional" by this architecture and which are not implemented by the platform must be ignored by the platform on a *Store* and must be returned as 0's on a *Load*, including the reserved or optional bits of a partially implemented field.

R1–2.12–2. Bits and bytes which are marked as "reserved" by this architecture must be ignored by the platform on a *Store* and must be returned as 0's on a *Load*, except that bits that are marked as "reserved" and which were previously defined by the architecture maybe be treated appropriately by legacy hardware (such bits in this architecture will state the value that software must use henceforth).

R1–2.12–3. Bits and bytes marked as "reserved" must be set to 0 by the OS on a *Store*, except as otherwise defined by the architecture, and must be ignored on a *Load*.

3 Address Map

The address map of an LoPAPR platform is made up of several distinct areas. These areas are one of five basic types. Each of these types has its own general characteristics such as coherency, alignment, size restrictions, variability of starting address and size, the system action on access of the area, and so on. This chapter gives details on some of those characteristics, and other chapters define the other characteristics. The variable characteristics of these areas are reported to the OS via properties in the OF device tree.

3.1 Address Areas

The following is a definition of the five areas and some of their characteristics:

- *System Memory* refers to memory which forms a coherency domain with respect to the PA processor(s) that execute application software on a system. See Section 5.2, “Memory Architecture,” on page 95 for details on aspects of coherence. *System Memory Spaces* refer to one or more pieces that together form the System Memory. System Memory areas may be marked with a special value of the **`"status"`** property of “reserved” which means that this memory is not for general use by the base OS, but may be reserved for use by OS extensions (see Section 5.2.8, “Reserved Memory,” on page 100). Some System Memory areas may be preservable across boots (see Section 2.1.3.6.12, “Persistent Memory and Memory Preservation Boot (Storage Preservation Option),” on page 49).
- *Peripheral Memory Space* refers to a range of real addresses which are assigned to the Memory Space of a Host Bridge (HB) or System Bus attached IOA, and which are sufficient to contain all of the *Load* and *Store* address space requirements of all IOAs in the Memory Space of the I/O bus that is generated by the HB or which are encompassed by the System Bus attached IOA. The frame buffer of a graphics IOA is an example of a device which may reside in the Peripheral Memory Space. Due to space limitations in the address space below 4 GB, the HBs of platforms may split this space into two pieces; one to support the IOAs that need to have their addresses below 4 GB (because they only support 32-bit addresses) and another to support the IOAs that can have their addresses above 4 GB (because they support 64-bit addresses). In addition to a Memory Space, many types of I/O buses have a separate address space called the I/O Space. An HB which generates such I/O buses must decode another address range, the Peripheral I/O Space.[1]
- *Peripheral I/O Space* refers to a range of real addresses which are assigned to the I/O Space of an HB or System Bus attached IOA and which are sufficient to contain all of the *Load* and *Store* address space requirements of all the IOAs in the I/O Space of the I/O bus that is generated by the HB or which are encompassed by the System Bus IOA. A keyboard controller is an example of an IOA which may require Peripheral I/O Space addresses.
- *System Control Area* (SCA) refers to a range of addresses which contains all reserved addresses (architected or unarchitected) which are not part of one of the other defined address spaces. For example, the system ROM(s), unarchitected platform-dependent addresses used by firmware and Run-Time Abstraction Services for control of the platform, and architected entities like interrupt controller addresses when those addresses are not in another defined address space.

1.A peripheral space may also include a “configuration” address space. The configuration space is abstracted by a Run-Time Abstraction Service (for example, see Section 7.3.4, “PCI Configuration Space,” on page 133).

- *Undefined* refers to areas that are not one of the above four areas. The result of accessing one of these areas is defined in Section 10.2.1.1, “Error Indication Mechanisms,” on page 284 as an invalid address error.

In addition to the above definitions, it is convenient, relative to I/O operations, to define a Partitionable Endpoint. A *Partitionable Endpoint* (PE) is an I/O subtree that can be treated as a unit for the purposes of partitioning and error recovery. A PE may be a single or multi-function IOA, a function of a multi-function IOA, or multiple IOAs (possibly including switch and bridge structures above the multiple IOAs). See Section 4.1, “I/O Topologies and Endpoint Partitioning,” on page 71 for more information about PEs.

In describing the characteristics of these various areas, it is convenient to have a nomenclature for the various boundary addresses. Table 5, “Map Legend,” on page 60 defines the labels which are used in this document when describing the various address ranges. Note that “bottom” refers to the smallest address of the range and “top” refers to the largest address.

Table 5. Map Legend

Label	Description
BIOn	Bottom of Peripheral I/O Space for HBn (n=0, 1, 2,...). The OF property **“ranges”** in the OF device tree for HBn contains the value of BIOn.
TIOn	Top of Peripheral I/O Space for HBn (n=0, 1, 2,...). The value of TIOn can be determined by adding the size of the area as found in the OF property **“ranges”** in the OF device tree for HBn to the value of BIOn found in that same property and then subtracting 1. This architecture allows at most one Peripheral I/O area per HB which may be above or below 4 GB. For any given n, BIOn to TIOn cannot span from the first 4 GB of address space to the second.
BPMn,m	Bottom of Peripheral Memory Space m (m=0,1) for HBn (n=0, 1, 2,...), as viewed from the system side of HBn. The OF property **“ranges”** in the OF device tree for HBn contains the value of BPMn,m.
BPM’n,m	Bottom of Peripheral Memory Space m (m=0,1) for HBn (n=0, 1, 2,...), as viewed from the I/O side of the HBn. That is, this is the value to which BPMn,m gets translated to as it passes through the HB. The OF property **“ranges”** in the OF device tree for HBn contains the value of BPM’n,m. BPM’n,m may be equal to BPMn,m or may not be.
TPMn,m	Top of Peripheral Memory Space m (m=0,1) for HBn (n=0, 1, 2,...) as viewed from the system side of HBn. The Peripheral Memory Space address range is in the OF device tree, as indicated by the **“ranges”** property in the node in the OF device tree for HBn; BPMn,m to TPMn,m. The value of TPMn,m can be determined by adding the size of the area as found in the OF property **“ranges”** in the OF device tree for HBn to the value of BPMn,m found in that same property and then subtracting 1. This architecture allows for one or two Peripheral Memory areas per HB (hence, m=0,1). A Peripheral Memory area may be above 4 GB or below. For any given n, BPMn,m to TPMn,m cannot span from the first 4 GB of address space to the second.
TPM’n,m	Top of Peripheral Memory Space m (m=0,1) for HBn (n=0, 1, 2,...) as viewed from the I/O side of HBn. The value of TPM’n,m can be calculated from the values in the **“ranges”** property as was TPMn,m. In some cases TPM’n,m is required to be equal to TPMn,m and in some cases it is not required to be equal. For any given n, BPM’n,m to TPM’n,m cannot span from the first 4 GB of address space to the second.
BSCAn	Bottom of System Control Area. Corresponding top of the System Control Area is TSCAn. This architecture allows for one or two SCAs per platform. The SCA below 4 GB is at the top (largest addresses) of the lower 4 GB range.
TSCAn	Top of System Control Area. For any given n, BSCAn to TSCAn cannot span from the first 4 GB of address space to the second.
BSMn	Bottom of System Memory Space n (n=0, 1, 2,...); BSM0 = 0. The OF property **“reg”** in the OF device tree for the Memory Controller’s node contains the value of BSMn.
TSMn	Top of System Memory Space n (n=0, 1, 2,...). The value of TSMn can be determined by adding the value of BSMn as found in the Memory Controller’s node of the OF device tree to the value of the size of that area as found in the same property, and then subtracting 1.

Table 5. Map Legend ***(Continued)***

Label	Description
BTTAn,m	Bottom of TCE Translatable Address space m (m=0, 1, 2,...) for HBn (n=0, 1, 2,...) as viewed from the I/O side of HBn. This is the bottom of an address range that is translatable by a Translation Control Entry (TCE) table. The value of BTTAn,m is obtained from the **"ibm,dma-window"** or **"ibm,my-dma-window"** property in the OF device tree.
TTTAn,m	Top of TCE Translatable Address space m (m=0, 1, 2,...) for HBn (n=0, 1, 2,...) as viewed from the I/O side of HBn. This is the top of an address range that is translatable by a TCE table. The range BTTAn,m to TTTAn,m is not accessible by more than one PE for any given "n". The value of TTTAn,m can be determined by adding the size of the area as found in the OF property **"ibm,dma-window"** or **"ibm,my-dma-window"** in the OF device tree for HBn to the value of BTTAn,m found in that same property and then subtracting 1.

The figures found in Section 3.2.3, "Example Address Maps," on page 67, show examples of the areas referenced by the labels in Table 5.

The OS and other software should not use fixed addresses for these various areas. A given platform may, however, make some of these addresses unchangeable. Each of these areas is defined in the OF device tree in the node of the appropriate controller. This gives platforms the most flexibility in implementing the System Address Map to meet their market requirements.

R2–3.1–1. All unavailable addresses in the Peripheral Memory and Peripheral I/O Spaces must be conveyed in the OF device tree.

a. A **"device_type"** of "reserved" must be used to specify areas which are not to be used by software and not otherwise reported by OF.

b. Shadow aliases must be communicated as specified by the appropriate OF bus binding.

R2–3.1–2. There must not be any address generated by the system which causes the system to hang.

Hardware Implementation Note: The reason for Requirement R2–3.1–1 is to reserve address space for registers used only by the firmware or addresses which are used only by the hardware.

3.2 Address Decoding (or Validating) and Translation

In general, different components in the hardware are going to decode the address ranges for the various areas. In some cases the component may be required to translate the address to a new address as it passes through the component. The requirements, below, describe the various system address decodes (or validating) and, where appropriate, what address transforms take place outside of the processor.

The HB requirements in this section refer to HBs which are defined by this architecture. Currently, there is only one HB defined by this architecture, and that is the PHB. HBs which implement I/O buses other than those defined by this architecture may or may not require changes to this addressing model.

The reader may want to reference the example address maps found in Section 3.2.3, "Example Address Maps," on page 67, while reading through the requirements of this section.

3.2.1 *Load* and *Store* Address Decoding and Translation

Load and *Store* operations may be targeted at System Memory or I/O. The latter is called Memory Mapped I/O (MMIO).

R2–3.2.1–1. Processor *Load* and *Store* operations must be routed and translated as shown in Table 6, "Processor Bus Address Space Decoding and Translation," on page 62.

Table 6. Processor Bus Address Space Decoding and Translation

Address Range at Processor Bus	Route and Translation Requirements	Other Requirements and Comments
BSCAn to TSCAn (n=0, 1)	To ROM controller or to a platform dependent area. Translation dependent on implementation.	Areas other than ROM are reserved for firmware use, or have their address passed by the OF device tree.
BIOn to TIOn (n=0,1, 2,...)	Send through the HB to the I/O space of the I/O bus, translating by subtracting the value of BIO from each address in this range (that is, translate BIO to TIO to be at 0 to (TIO - BIO) on the I/O side).	
BPMn,m to TPMn,m (n=0, 1, 2,...) (m=0, 1)	Send through HBn to the Memory Space of the I/O bus. ◆ If BPMn,m < 4 GB, do not translate an address in the BPMn,m to TPMn,m range as the transaction passes through the bridge (that is, BPM'n,m = BPMn,m and TPM'n,m = TPMn,m). ◆ If BPMn,m is at or above 4 GB then if BPM'n,m is to be below 4 GB (for 32-bit IOAs) then translate addresses in the BPMn,m to TPMn,m range so that this address range becomes BPM'n,m to TPM'n,m (where BPM'n,m and TPM'n,m are less than 4 GB) as the transaction passes through the bridge, otherwise do not translate an address in the BPMn,m to TPMn,m range as the transaction passes through the bridge (for 64-bit IOAs which are configured at or above 4 GB).	Platforms that need to support both 32-bit capable and 64-bit capable IOAs and do not want to configure the 64-bit capable IOAs below 4 GB need to support two Peripheral Memory spaces per HB.
BSMm to TSMm (m>0)	To System Memory Space m, no translation.	Can be at or above 4 GB, or below BSCA0.
0 to TSM0	To System Memory Space 0, no translation	
All other addresses	See Section 10.2.1.1, "Error Indication Mechanisms," on page 284.	Access is to undefined space.

R2–3.2.1–2. There must be no architected address spaces (Peripheral Memory, Peripheral I/O, SCA, or System Memory) which span the (4GB - 1) to 4 GB boundary.

R2–3.2.1–3. The following are the System Control Area requirements:

a. The platform must have at most one System Control Area below 4 GB and at most one per platform or per NUMA node at or above 4 GB.

b. The System Control Area must not overlap with the System Memory Space(s), Peripheral Memory Space(s), or the Peripheral I/O Space(s) in the platform.

R2–3.2.1–4. The following are the System Memory Space requirements:

a. Each platform must have at least one System Memory Space.

b. The System Memory Space(s) must not overlap with the Peripheral I/O Space(s), Peripheral Memory Space(s), the System Control Area, or other System Memory Space(s) in the platform.

c. The first System Memory Space must start at address 0 (BSM0 = 0), must be at least 128 MB before a second System Memory Space is added and must be contiguous.

d. Each of the additional (optional) System Memory Space(s) must start on a 4 KB boundary.

e. Each of the additional (optional) System Memory Space(s) must be contiguous within itself.

f. There must be at most eight System Memory Spaces below BSCA0 and at most eight at or above 4 GB.

g. If multiple System Memory Spaces exist below 4 GB, then they must not have any Peripheral Memory or Peripheral I/O Spaces interspersed between them and if multiple System Memory Spaces exist above 4

GB, then they must not have any Peripheral Memory or Peripheral I/O Spaces interspersed between them.

R2–3.2.1–5. The following are the Peripheral Memory Space requirements:

a. The Peripheral Memory Space(s) must not overlap with the System Memory Space(s), Peripheral I/O Space(s), the System Control Area, or other Peripheral Memory Space(s) in the platform.

b. The size of each Peripheral Memory Space (TPMn,m - BPMn,m + 1) must be a power of two for sizes up to and including 256 MB, with the minimum size being 1 MB, and an integer multiple of 256 MB plus a power of two which is greater than or equal to 1 MB for sizes greater than 256 MB (for example, 1 MB, 2 MB, 4 MB, 8 MB, 16 MB, 32 MB, 64 MB, 128 MB, 256 MB, (256 + 1) MB, (256 + 2) MB,..., (512 + 1) MB,...).

c. The boundary alignment for each Peripheral Memory Space must be an integer multiple of the size of the space up to and including 256 MB and must be an integer multiple of 256 MB for sizes greater than 256 MB.

d. There must be at most two Peripheral Memory Spaces per HB.

e. If the Peripheral Memory Space for a HB is below 4 GB, then the address must not be translated as it passes through the HB from the system side to the I/O side of the HB (see Table 6, “Processor Bus Address Space Decoding and Translation,” on page 62).

f. If the Peripheral Memory Space for a HB is above 4 GB, then the address may or may not be translated as it passes through the HB from the system side to the I/O side of the HB, but if it is translated, then the translated address range must be aligned on a boundary which is an integer multiple of the size of the Peripheral Memory Space.

Implementation Note: Relative to Requirement R2–3.2.1–5f, not all OSs can support BPM’ to TPM’ being above 4 GB.

R2–3.2.1–6. The following are the Peripheral I/O Space requirements:

a. The Peripheral I/O Space(s) must not overlap with the System Memory Space(s), Peripheral Memory Space(s), the System Control Area, or other Peripheral I/O Space(s) in the platform.

b. The size of each Peripheral I/O Space (TIOn - BIOn + 1) must be a power of two with the minimum size being 64 KB (that is, sizes of 64 KB, 128 KB, 256 KB, 512 KB, 1 MB, 2 MB, 4 MB, 8 MB, 16 MB, 32 MB, 64 MB, and so on, are acceptable).

c. The boundary alignment for each Peripheral I/O Space must be an integer multiple of the size of the space.

d. There must be at most one Peripheral I/O Space per HB.

R2–3.2.1–7. All System Memory must be accessible via DMA operation from all IOAs in the system, except where LPAR requirements limit accessibility of an IOA belonging to one partition to the System Memory of another partition.

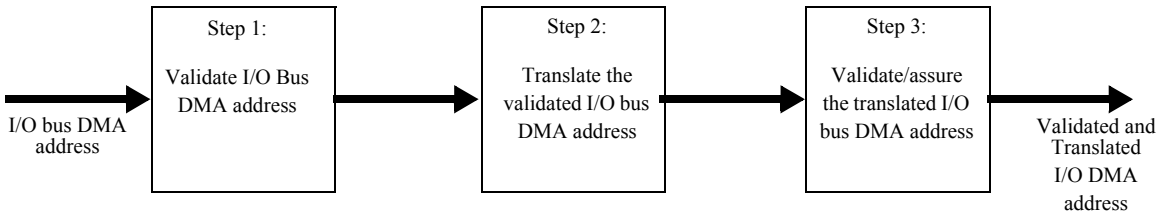
Hardware Implementation Notes: Memory controller and memory card designers who are designing for 64-bit platforms should be careful to consider that the amount of I/O space below 4 GB is reduced by the amount of System Memory space below 4 GB. Therefore it may be prudent to design the hardware to allow minimization of the amount of System Memory below 4 GB, in order to allow maximization of the space for 32-bit Peripheral Memory and Peripheral I/O spaces below 4 GB.

The beginning addresses and sizes of the Peripheral I/O Space(s) and Peripheral Memory Space(s), are controlled by firmware. Information about the address map is reported by the OF Device Tree or, for items that can change, through RTAS calls (for example, for Dynamic Reconfiguration, through the *ibm,configure-connector* RTAS call).

Certain System Memory addresses must be reserved in all systems for specific uses (see Section 5.1.2, “PA Processor Differences,” on page 93 and Section 5.2.8, “Reserved Memory,” on page 100 for more information).

3.2.2 DMA Address Validation and Translation

Figure 5, “PE DMA Address Validation and Translation in the Platform,” on page 64 is a representation of how the validation and translation mechanism works, along with a description of the steps which are involved. At the core of the translation mechanism is the Translation and Control Entry (TCE) table.

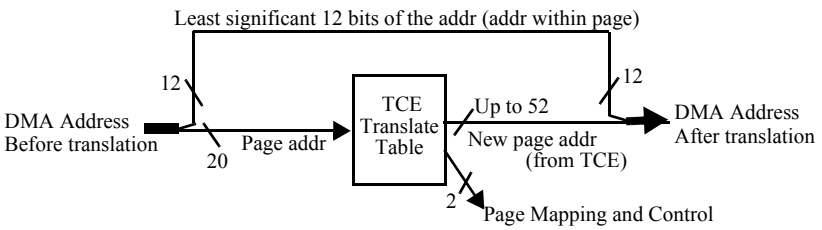


Step 1 includes:

- Validate that the PE is allowed to use that address, including that the address is not outside of the translatable range for the PE (Note 1).

Step 2 includes:

- Lookup the TCE for the transaction, using the appropriate TCE table for the PE, using the high order (page address) bits of the DMA address, as the index into the table.
- Validate that the Page Mapping and Control bits in the TCE accessed, indicate a valid TCE and that the type of operation matches the bits, otherwise signal an invalid address error (See Section 10.2.1.1, “Error Indication Mechanisms,” on page 284),
- Replace the high-order (page address) bits of the DMA address with the Real Page Number (RPN) bits from the TCE. The number of low order bits used from the original DMA address in the new translated address is 12 bits for a 4 KB I/O page size. This is shown, below, for a 32-bit to 64-bit address translation with 4 KB I/O page size.



Step 3 includes:

- Validate or assure that the translated address does not re-access the same HB or another HB (Note 1). If the firmware controls the TCE content and the address is translated by a TCE in Step 2, then the firmware can assure this.
- Do any other validation required by the platform (Note 1).

Notes:

1. For the types of errors signalled on validation or translation failures, see the requirements detailed in this chapter.

Figure 5. PE DMA Address Validation and Translation in the Platform

3.2.2.1 DMA Addressing Requirements

R2–3.2.2.1–1. Upon receiving a DMA transaction to the Memory Space of an I/O bus, the HB must perform the validation and translation steps, as indicated in Figure 5, “PE DMA Address Validation and Translation in the Platform,” on page 64 and in Table 7, “DMA Address Decoding and Translation (I/O Bus Memory Space),” on page 65.

Table 7. DMA Address Decoding and Translation (I/O Bus Memory Space)

Address Range at I/O Side of HBn	Route and Translation Requirements	Other Requirements and Comments
BPM’n,m to TPM’n,m (n=0, 1, 2,...) (m=0, 1) (note 1)	HB does not respond or responds and signals an invalid address error (See Section 10.2.1.1, “Error Indication Mechanisms,” on page 284).	
BTTAn,m to TTTAn,m (n=0, 1, 2,...) (m=0, 1, 2,...) (note 1)	If the PE that is trying to access this space is allowed to access this space, then translate via the TCE table (as specified in Section 3.2.2.2, “DMA Address Translation and Control via the TCE Mechanism,” on page 65) and pass the translated address through the HB, otherwise generate an invalid address or TCE extent error, as appropriate (See Section 10.2.1.1, “Error Indication Mechanisms,” on page 284).	See Notes 2, 3
All other addresses	Generate an invalid address error (See Section 10.2.1.1, “Error Indication Mechanisms,” on page 284).	See Note 3

Notes:
1. n = # of HB Viewing or Receiving the Operation, m = # of instance within the HB.
2. After translation of the address, if the translated address would re-access the same HB or another HB (for example, is in the Peripheral Memory Space or Peripheral I/O Space of that HB or another HB), then the HB generates an invalid address error (See Section 10.2.1.1, “Error Indication Mechanisms,” on page 284).
3. If the Enhanced I/O Error Handling (EEH) option is implemented and enabled, then on an error, the PE will enter the DMA Stopped State (See Section 4.4.1, “Enhanced I/O Error Handling (EEH) Option,” on page 85).

R2–3.2.2.1–2. An HB must not act as a target for operations in the I/O Space of an I/O bus.

3.2.2.2 DMA Address Translation and Control via the TCE Mechanism

This architecture defines a Translation and Control Entry (TCE) mechanism for translating and controlling DMA addresses. There are several reasons for doing such translations, including:

- To provide a mechanism for increasing the number of addressing bits for some IOAs. For example, IOAs which are only capable of accessing up to 4 GB via DMA need a way to access above that limit when used in 64-bit addressing systems and the addressing requirements go beyond 4 GB.
- To provide a redirection mechanism. A redirection mechanism is needed, even for 64-bit addressing capable IOAs, in order to provide the protection and indirection benefits provided by such a translation.

The description of how the access to the TCE table occurs, for the translation of a 32-bit address and using a 4 KB I/O page size, follows. The most significant 20 bits of the address (for example, AD[31:12], for PCI) is used as an offset into the TCE table for the PE to select the TCE. Thus, the first TCE maps the addresses BTTAn to BTTAn + 0x00000FFF of the Memory Space of the I/O bus; the second entry controls translation of addresses BTTAn + 0x00001000 to BTTAn + 0x00001FFF, and so on. The translated real system address is generated as follows. The Real Page Number (RPN) from the TCE replaces the 20 most significant bits of the address from the I/O bus. The least significant 12 bits from the I/O bus address are used as-is for the least significant 12 bits of the new address.

Thus, the TCE table entries have a one-to-one correspondence with the first *n* pages of the Memory Space of the I/O bus starting at BTTAn that corresponds to the TCE table. The size of the Memory address space of the I/O bus that can

be mapped to the system address space for a particular HB depends on how much System Memory is allocated to the TCE table(s) and on how much mappable I/O bus Memory Space is unavailable due to IOAs which are mapped there.

Each TCE also contains two control bits. These are used to identify whether that page is mapped to the system address space, and if the page is mapped, whether it is mapped read/write, read only, or write only. See the Table 8, “TCE Definition,” on page 66 for a definition of these control bits.

The TCE table is the analogue of the system translation tables. However, unlike the system translation tables, the dynamic page faulting of memory during an I/O operation is not required (the page fault value, 0b00, in the TCE Page Mapping and Control field is used for error detection; that is, access to an invalid TCE by the I/O creates an error indication to the software).

The size and location of the HB’s TCE table is set up and changed only by the firmware.

R2–3.2.2.2–1. The platform must provide the **`“64-bit-addressing”`** and **`“ibm,extended-address”`** OF properties in all HB nodes of the device tree and the **`“ibm,extended-address”`** OF property in the **`root`** node of the OF device tree.

R2–3.2.2.2–2. The bits of the TCE must be implemented as defined in Table 8, “TCE Definition,” on page 66.

Table 8. TCE Definition

Bits	Description
0 to 51	RPN: If the page mapping and control field of the TCE indicate anything other than page fault, then these bits contain the Real Page Number (RPN) to which the bus address is mapped in the system address space. In certain HB implementations, all of these bits may not be required, however enough bits must be implemented to match the largest real address in the platform.
52 to 61	Reserved for future use.
62 to 63	Page Mapping and Control: These bits define page mapping and read-write authority. They are coded as follows: 00 Page fault (no access) 01 System address space (read only) 10 System address space (write only) 11 System address space (read/write) Code point 0b00 signifies that the page is not mapped. It must be used to indicate a page fault error. Hardware must not change its state based on the value in the remaining bits of a TCE when code point 0b00 is set in this field of the TCE. For accesses to system address space with an invalid operation (write to a read-only page or read to a write-only page), the HB generates an error. See Section 10.2.1.1, “Error Indication Mechanisms,” on page 284 for more information about error handling.

R2–3.2.2.2–3. If the address that the HB would use to access the TCE table (in order to get the TCE) would access outside of the TCE table, then the HB must create a TCE extent error (See Section 10.2.1.1, “Error Indication Mechanisms,” on page 284).

R2–3.2.2.2–4. Enough bits must be implemented in the TCE so that DMA IOAs are able to access all System Memory addresses.

R2–3.2.2.2–5. Each PE must have its own independent TCE table.

R2–3.2.2.2–6. Any non-recoverable error while an HB is accessing its TCE table must result in a TCE access error; the action to be taken by the HB being defined under the TCE access error in Section 10.2.1.1, “Error Indication Mechanisms,” on page 284.

R2–3.2.2.2–7. In implementations which cache TCEs, if software changes a TCE, then the platform must perform the following steps: First, if any data associated with the page represented by that TCE is in an I/O bridge cache or buffer, the hardware must write the data, if modified, to System Memory. Secondly, it must invalidate the data in the cache. Finally, it must invalidate the TCE in the cache.

R2–3.2.2.2–8. Neither an IOA nor an HB must ever modify a TCE.

R2–3.2.2.2–9. If the page mapping and control bits in the TCE are set to 0b00, the hardware must not change its state based on the values of the remaining bits of the TCE.

R2–3.2.2.2–10. The OS must initialize all its TCEs upon receiving control from the platform.

3.2.3 Example Address Maps

Figure 6, “Example Address Map: One PHB, Peripheral Memory and Peripheral I/O Spaces below 4 GB,” on page 68 shows how to construct a simple address map with one PHB and with Peripheral Memory, Peripheral I/O, and SCA spaces below 4 GB.

Figure 7, “Example Address Map: Four PHBs, all Peripheral Memory and Peripheral I/O Spaces above 4GB,” on page 69 shows how to construct the address map with Peripheral Memory, Peripheral I/O, and SCA spaces above 4 GB. This configuration allows some overlap of the System Memory space and 32-bit I/O bus memory space (with the resulting loss of the TCE table in the overlap), while moving some of the SCA spaces above 4 GB. Several things can be noted from this configuration:

- I/O bus memory areas can overlap System Memory addresses (see memory space of PHB0). However, significant overlap of these I/O bus memory areas and the TCE table may significantly reduce the amount of TCE table space that is available for mapping I/O memory space to system address space (a potential performance impact).
- The System Memory which is above 4GB is shown starting at 4GB. This architecture also allows this to be pushed further up, with Peripheral Memory, Peripheral I/O, and SCAs existing above 4 GB and below the System Memory areas.
- BPM’n,m to TPM’n,m spaces for different PHBs (different “n”) are allowed to occur at the same memory addresses in the various memory spaces of different I/O buses, but are not required to do so (and are not shown as the same in the figure). Implementations are likely have BPM’n,m to TPM’n,m at the same address range for all “n” when the BPM’n,m to TPM’n,m ranges are below 4 GB.

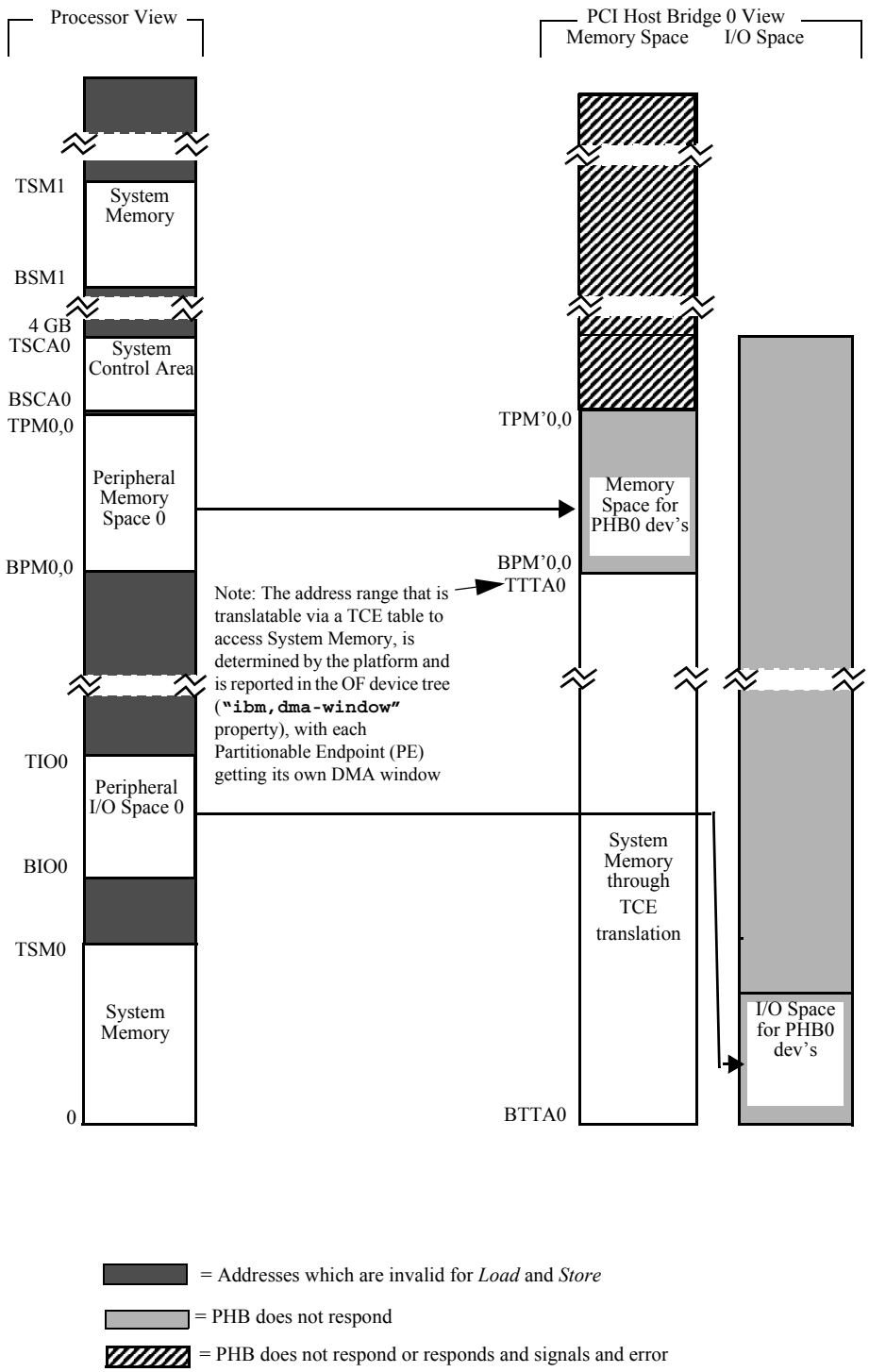
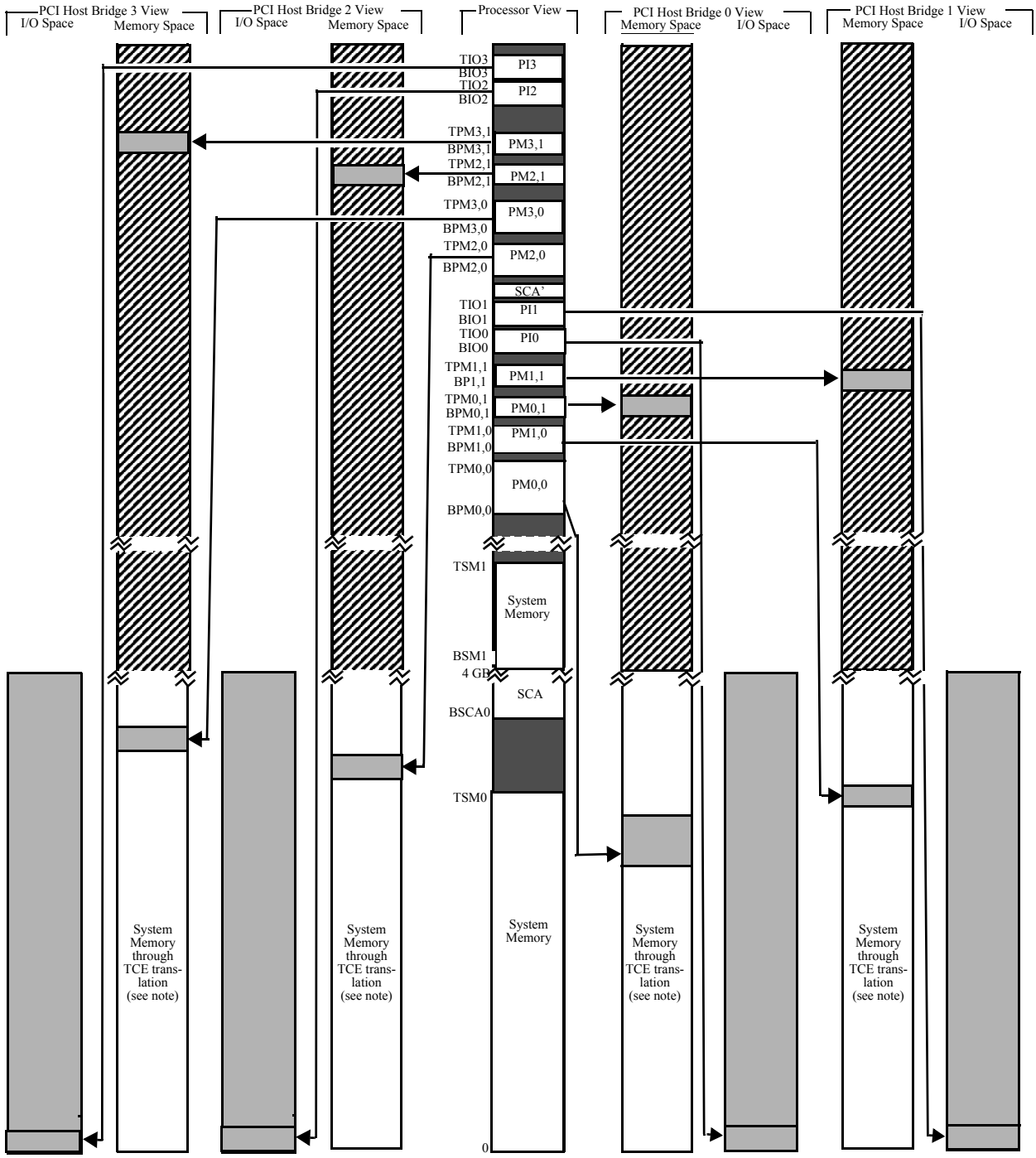


Figure 6. Example Address Map: One PHB, Peripheral Memory and Peripheral I/O Spaces below 4 GB



Note: The address range that is translatable via a TCE table to access System Memory, is determined by the platform and is reported in the OF device tree (**`"ibm,dma-window"`** property), with each Partitionable Endpoint (PE) getting its own DMA window

Legend: = Addresses which are invalid for *Load* and *Store*; = PHB does not respond
PM = Peripheral Memory space, PI = Peripheral I/O space; = PHB does not respond or responds and signals an error

Figure 7. Example Address Map: Four PHBs, all Peripheral Memory and Peripheral I/O Spaces above 4GB

4 I/O Bridges and Topologies

There will be at least one bridge in a platform which interfaces to the system interconnect on the processor side, and interfaces to the Peripheral Component Interface (PCI) bus on the other. This bridge is called the PCI Host Bridge (PHB). The architectural requirements on the PHB, as well as other aspects of the I/O structures, PCI bridges, and PCI Express switches are defined in this chapter.

4.1 I/O Topologies and Endpoint Partitioning

As systems get more sophisticated, partitioning of various components of the system will be used, in order to obtain greater Reliability, Availability, and Serviceability (RAS). For example, Dynamic Reconfiguration (DR) allows the removal, addition, and replacement of components from an OS's pool of resources, without having to stop the operation of that OS. In addition, Logical Partitioning (LPAR) allows the isolation of resources used by one OS from those used by another. This section will discuss aspects of the partitioning of the I/O subsystem. Further information on DR and LPAR can be found in Chapter 13, "Dynamic Reconfiguration (DR) Architecture," on page 355 and Chapter 14, "Logical Partitioning Option," on page 385.

To be useful, the granularity of assignment of I/O resources to an OS needs to be fairly fine-grained. For example, it is not generally acceptable to require assignment of all I/O under the same PCI Host Bridge (PHB) to the same partition in an LPARed system, as that restricts configurability of the system, including the capability to dynamically move resources between partitions[1]. To be able to partition I/O adapters (IOAs), groups of IOAs or portions of IOAs for DR or to different OSs for LPAR will generally require some extra functionality in the platform (for example, I/O bridges and firmware) in order to be able to partition the resources of these groups, or endpoints, while at the same time preventing any of these endpoints from affecting another endpoint or getting access to another endpoint's resources. These endpoints (that is, I/O subtrees) that can be treated as a unit for the purposes of partitioning and error recovery will be called Partitionable Endpoints (PEs)[2] and this concept will be called Endpoint Partitioning.

A PE is defined by its Enhanced I/O Error Handling (EEH) domain and associated resources. The resources that need to be partitioned and not overlap with other PE domains include:

- The Memory Mapped I/O (MMIO) *Load* and *Store* address space which is available to the PE. This is accomplished by using the processor's Page Table mechanism (through control of the contents of the Page Table Entries) and not having any part of two separate PEs' MMIO address space overlap into the same 4 KB system page. Additionally, for LPAR environments, the Page Table Entries are controlled by the hypervisor.
- The DMA I/O bus address space which is available to the PE. This is accomplished by a hardware mechanism (in a bridge in the platform) which enforces the correct DMA addresses, and for LPAR, this hardware enforcement is set up by the hypervisor. It is also important that a mechanism be provided for LPAR such that the I/O bus addresses can further be limited at the system level to not intersect; so that one PE cannot get access to a partition's memory to which it should not have access. The Translation Control Entry (TCE) mechanism, when controlled by the firmware

1.Dynamic LPAR or DLPAR is defined by the Logical Resource Dynamic Reconfiguration (LRDR) option. See Section 13.7, "Logical Resource Dynamic Reconfiguration (LRDR)," on page 377 for more information. Assignment of all IOAs under the same PHB to one partition may be acceptable if that I/O is shared via the Virtual I/O (VIO) capability defined in Chapter 17, "Virtualized Input/Output," on page 597.

2.A "Partitionable Endpoint" in this architecture is not to be confused with what the PCI Express defines as an "endpoint." PCI Express defines an endpoint as "a device with a Type 0x00 Configuration Space header." That means PCI Express defines any entity with a unique Bus/Dev/Func # as an endpoint. In most implementations, a PE will not exactly correspond to this unit.

(for example, a hypervisor), is such a mechanism. See Section 3.2.2.2, “DMA Address Translation and Control via the TCE Mechanism,” on page 65 for more information on the TCE mechanism.

- The configuration address space of the PE, as it is made available to the device driver. This is accomplished through controlling access to a PE’s configuration spaces through Run Time Abstraction Services (RTAS) calls, and for LPAR, these accesses are controlled by the hypervisor.
- The interrupts which are accessible to the PE. An interrupt cannot be shared between two PEs. For LPAR environments, the interrupt presentation and management is via the hypervisor.
- The error domains of the PE; that is, the error containment must be such that a PE error cannot affect another PE or, for LPAR, another partition or OS image to which the PE is not given access. This is accomplished though the use of the Enhanced I/O Error Handling (EEH) option of this architecture. For LPAR environments, the control of EEH is through the hypervisor via several RTAS calls.
- The reset domain: A reset domain contains all the components of a PE. The reset is provided programmatically and is intended to be implemented via an architected (non implementation dependent) method.[1] Resetting a component is sometimes necessary in order to be able to recover from some types of errors. A PE will equate to a reset domain, such that the entire PE can be reset by the *ibm,set-slot-reset* RTAS call. For LPAR, the control of the reset from the RTAS call is through the hypervisor.

In addition to the above PE requirements, there may be other requirements on the power domains. Specifically, if a PE is going to participate in DR, including DLPAR,[2] then either the power domain of the PE is required to be in a power domain which is separate from other PEs (that is, power domain, reset domain, and PE domain all the same), or else the control of that power domain and PCI Hot Plug (when implemented) of the contained PEs will be via the platform or a trusted platform agent. When the control of power for PCI Hot Plug is via the OS, then for LPAR environments, the control is also supervised via the hypervisor.

It is possible to allow several cooperating device drivers to share a PE. Sharing of a PE between device drivers within one OS image is supported by the constructs in this architecture. Sharing between device drivers in different partitions is beyond the scope of the current architecture.

A PE domain is defined by its top-most (closest to the PHB) PCI configuration address (in the terms of the RTAS calls, the *PHB_Unit_ID_Hi, PHB_Unit_ID_Low, and config_addr*), which will be called the *PE configuration address* in this architecture, and encompasses everything below that in the I/O tree. The top-most PCI bus of the PE will be called the *PE primary bus*. Determination of the PE configuration address is made as described in Table 9, “Conventional PCI Express PE Support Summary,” on page 72.

A summary of PE support can be found in Table 9, “Conventional PCI Express PE Support Summary,” on page 72. This architecture assumes that there is a single level of bridge within a PE if the PE is heterogeneous (some Conventional PCI Express), and these cases are shown by the shaded cells in the table.

Table 9. Conventional PCI Express PE Support Summary

Function	IOA Type	PE Primary Bus PCI Express
PE determination (is EEH supported for the IOA?)	All	Use the *ibm,read-slot-reset-state2* RTAS call.

1.For example, through a Standard Hot Plug Controller in a bridge, or through the Secondary Bus Reset bit in the Bridge Control register of a PCI bridge or switch.

2.To prevent data from being transferred from one partition to another via data remaining in an IOA’s memory, most implementations of DLPAR will require the power cycling of the PE after removal from one partition and prior to assigning it to another partition.

Table 9. Conventional PCI Express PE Support Summary ***(Continued)***

Function	IOA Type	PE Primary Bus PCI Express
PE reset	All	PE reset is required for all PEs and is activated/deactivated via the *ibm,set-slot-reset* RTAS call. The PCI configuration address used in this call is the PE configuration address (the reset domain is the same as the PE domain).
ibm,get-config-addr-info2 RTAS call	PCI Express	Required to be implemented.
Top of PE domain determination[a] (How to obtain the PE configuration address)	PCI Express	Use the *ibm,get-config-addr-info2* RTAS call to obtain PE configuration address.
Shared PE determination[b] (is there more than one IOA per PE?)	PCI Express	Use the *ibm,get-config-addr-info2* RTAS call.
PEs per PCI Hot Plug domain and PCI Hot Plug control point	PCI Express	May have more than one PE per PCI Hot Plug DR entity, but a PE will be entirely encompassed by the PCI Hot Plug power domain. If more than one PE per DR entity, then PCI Hot Plug control is via the platform or some trusted platform agent.

a. PE configuration address is used as input to the RTAS calls which are used for PE control, namely: *ibm,set-slot-reset, ibm,set-eeh-option, ibm,slot-error-detail, ibm,configure-bridge*
b. If device driver is written for the shared PE environment, then this may be a don't care.

R1–4.1–1. All platforms must implement the *ibm,get-config-addr-info2* RTAS call.

R1–4.1–2. All platforms must implement the *ibm,read-slot-reset-state2* RTAS call.

R1–4.1–3. For the EEH option: The resources of one PE must not overlap the resources of another PE, including:

- Error domains
- MMIO address ranges
- I/O bus DMA address ranges (when PEs are below the same PHB)
- Configuration space
- Interrupts

R1–4.1–4. For the EEH option: All the following must be true relative to a PE:

a. An IOA function must be totally encompassed by a PE.

b. All PEs must be independently resetable by a reset domain.

Architecture Note: The partitioning of PEs down to a single IOA function within a multi-function IOA requires a way to reset an individual IOA function within a multi-function IOA. For PCI, the only mechanism defined to do this is the optional PCI Express Function Level Reset (FLR). A platform supports FLR if it supports PCI Express and the partitioning of PEs down to a single IOA function within a multi-function IOA. When FLR is supported, if the *ibm,set-slot-reset* RTAS call uses FLR for the *Function* 1/*Function* 0 (activate/deactivate reset) sequence for an IOA function, then the platform provides the **`"ibm,pe-reset-is-flr"`** property in the function's node of the OF device tree, See Section 7.3.11.2, "ibm,set-slot-reset," on page 182 for more information.

R1–4.1–5. The platform must own (be responsible for) any error recovery for errors that occur outside of all PEs (for example in switches and bridges above defined PEs).

Implementation Note: As part of the error recovery of Requirement R1–4.1–5, the platform may, as part of the error handling of those errors, establish an equivalent EEH error state in the EEH domains of all PEs below the error point, in order to recover the hardware above those EEH domains from its error state. The platform also returns a *PE Reset State* of 5 (PE is unavailable) with a *PE Unavailable Info* non-zero (temporarily unavailable) while a recovery is in progress.

R1–4.1–6. The platform must own (be responsible for) fault isolation for all errors that occur in the I/O fabric (that is, down to the IOA; including errors that occur on that part of the I/O fabric which is within a PE's domain).

R1–4.1–7. For the EEH option with the PCI Hot Plug option: All of the following must be true:

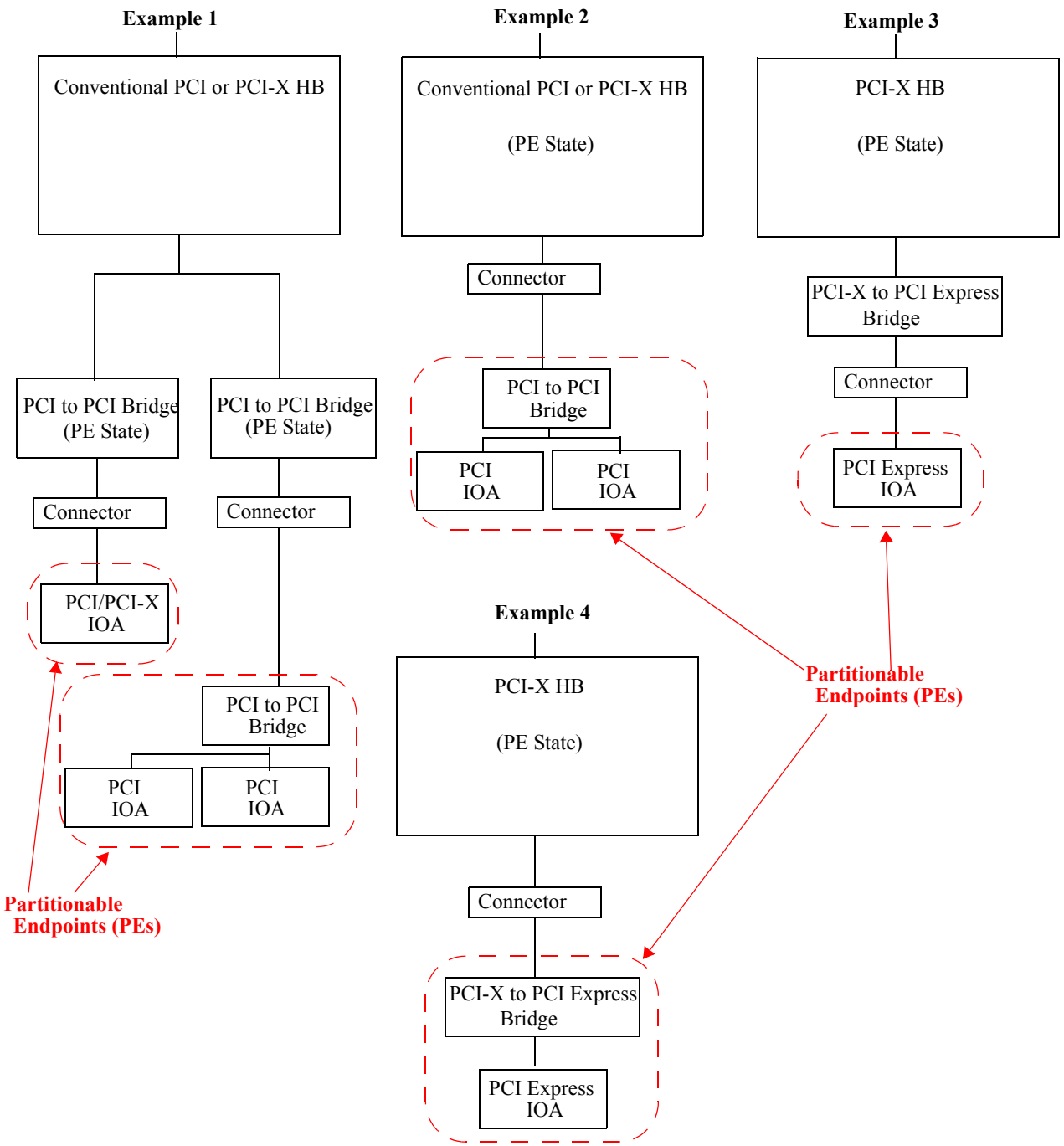
- If PCI Hot Plug operations are to be controlled by the OS to which the PE is assigned, then the PE domain for the PCI Hot Plug entity and the PCI Hot Plug power domain must be the same.
- All PE domains must be totally encompassed by their respective PCI Hot Plug power domain, regardless of the entity that controls the PCI Hot Plug operation.

R1–4.1–8. All platforms that implement the EEH option must enable that option by default for all PEs.

Implementation Notes:

1. See Requirement R1–14.3–1 and Requirement R1–14.3–2 relative to EEH requirements with LPAR.
2. Defaulting to EEH enabled, as required by Requirement R1–4.1–8 does not imply that the platform has no responsibility in assuring that all device drivers are EEH enabled or EEH safe before allowing their the Bus Master, Memory Space or I/O Space bits in the PCI configuration Command register of their IOA to be set to a 1. Furthermore, even though a platform defaults its EEH option as enabled, as required by Requirement R1–4.1–8 does not imply that the platform cannot disable EEH for a PE. See Requirement R1–4.4.1.1–18 for more information.

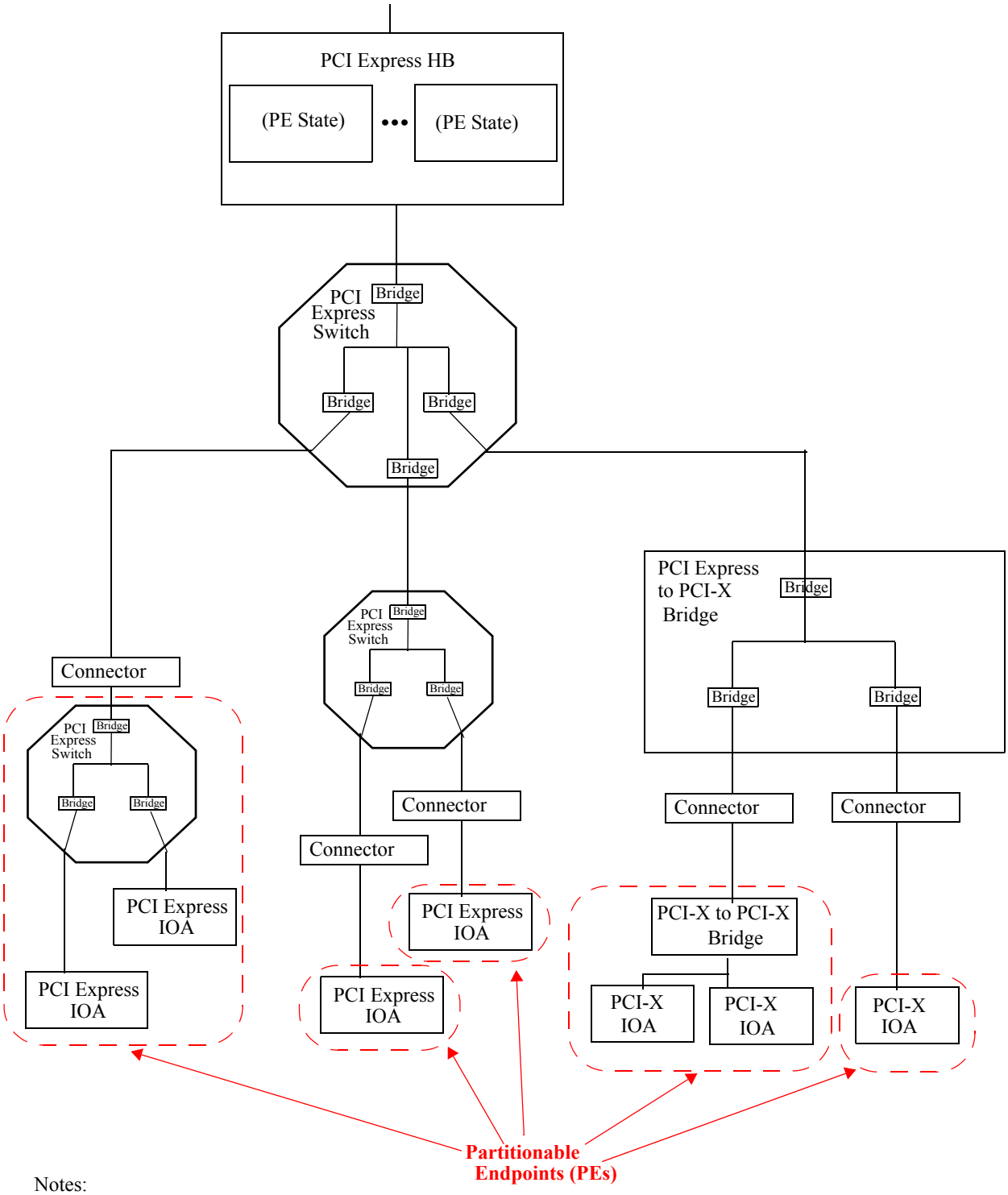
The following two figures show some examples of the concept of Endpoint Partitioning. See also Section 4.4.1, "Enhanced I/O Error Handling (EEH) Option," on page 85 for more information on the EEH option.



Notes:

1. Connectors are shown, but the connectors may or may not exist for a given implementation (connectors are meant to designate physical PCI Hot Plug connectors)
2. PE State includes such items as: EEH enablement state, MMIO Stopped state, and DMA Stopped state

Figure 8. PE and DR Partitioning Examples for Conventional PCI and PCI-X HBs.



Notes:

1. Connectors are shown, but the connectors may or may not exist for a given implementation (connectors are meant to designate physical PCI Hot Plug connectors)
2. PE State includes such items as: EEH enablement state, MMIO Stopped state, and DMA Stopped state

Figure 9. PE and DR Partitioning Examples for PCI Express HBs

4.2 PCI Host Bridge (PHB) Architecture

The PHB architecture places certain requirements on PHBs. There should be no conflict between this document and the PCI specifications, but if there is, the PCI documentation takes precedence. The intent of this architecture is to provide a base architectural level which supports the PCI architecture and to provide optional constructs which allow for use of 32-bit PCI IOAs in platforms with greater than 4 GB of system addressability.

R1–4.2–1. All PHBs that implement conventional PCI must be compliant with the most recent version of the *PCI Local Bus Specification* [18] at the time of their design, including any approved Engineering Change Requests (ECRs) against that document.

R1–4.2–2. All PHBs that implement PCI-X must be compliant with the most recent version of the *PCI-X Protocol Addendum to the PCI Local Bus Specification* [21] at the time of their design, including any approved Engineering Change Requests (ECRs) against that document.

R1–4.2–3. All PHBs that implement PCI Express must be compliant with the most recent version of the *PCI Express Base Specification* [22] at the time of their design, including any approved Engineering Change Requests (ECRs) against that document.

R1–4.2–4. All requirements defined in Chapter 3, “Address Map,” on page 59 for HBs must be implemented by all PHBs in the platform.

4.2.1 PHB Implementation Options

There are a few implementation options when it comes to implementing a PHB. Some of these become requirements, depending on the characteristics of the system for which the PHB is being designed. The options affecting PHBs, include the following:

- The Enhanced I/O Error Handling (EEH) option enhances RAS characteristics of the I/O and allows for smaller granularities of I/O assignments to partitions in an LPAR environment.
- The Error Injection (ERRINJCT) option enhances the testing of the I/O error recovery code. This option is required of bridges which implement the EEH option.

R1–4.2.1–1. All PHBs for use in platforms which implement LPAR must support EEH, in support of Requirements R1–14.3–1 and R1–14.3–2.

R1–4.2.1–2. All PCI HBs designed for use in platforms which will support PCI Express must support the PCI extended configuration address space and the MSI option.

4.2.2 PCI Data Buffering and Instruction Queuing

Some PHB implementations may include buffers or queues for DMA, *Load*, and *Store* operations. These buffers are required to be transparent to the software with only certain exceptions, as noted in this section.

Most processor accesses to System Memory go through the processor data cache. When sharing System Memory with IOAs, hardware must maintain consistency with the processor data cache and the System Memory, as defined by the requirements in Section 5.2, “Memory Architecture,” on page 95.

R1–4.2.2–1. PHB implementations which include buffers or queues for DMA, *Load*, and *Store* operations must make sure that these are transparent to the software, with a few exceptions which are allowed by the PCI architecture, by *Power ISA* [1], and by Section 5.2, “Memory Architecture,” on page 95.

R1–4.2.2–2. PHBs must accept up to a 128 byte MMIO *Loads*, and must do so without compromising performance of other operations.

4.2.2.1 PCI *Load* and *Store* Ordering

For the platform *Load* and *Store* ordering requirements, see Section 5.2.2, "Memory Mapped I/O (MMIO) and DMA Operations," on page 97 and the appropriate PCI specifications (per Requirements R1–4.2–1, R1–4.2–2, and R1–4.2–3). Those requirements will, for most implementations, require strong ordering (single threading) of all *Load* and *Store* operations through the PHB, regardless of the address space on the PCI bus to which they are targeted. Single threading through the PHB means that processing a *Load* requires that the PHB wait on the *Load* response data of a *Load* issued on the PCI bus prior to issuing the next *Load* or *Store* on the PCI bus.

4.2.2.2 PCI DMA Ordering

For the platform DMA ordering requirements, see the requirements in this section, in Section 5.2.2, "Memory Mapped I/O (MMIO) and DMA Operations," on page 97, and the appropriate PCI specifications (per Requirements R1–4.2–1, R1–4.2–2, and R1–4.2–3).

In general, the ordering for DMA path operations from the I/O bus to the processor side of the PHB is independent from the *Load* and *Store* path, with the exception stated in Requirement R1–4.2.2.2–6. Note that in the requirement, below, a *read request* is the initial request to the PHB and the *read completion* is the data phase of the transaction (that is, the data is returned).

R1–4.2.2.2–1. *(Requirement Number Reserved For Compatibility)*

R1–4.2.2.2–2. *(Requirement Number Reserved For Compatibility)*

R1–4.2.2.2–3. *(Requirement Number Reserved For Compatibility)*

R1–4.2.2.2–4. The hardware must make sure that a DMA read request from an IOA that specifies any byte address that has been written by a previous DMA write operation (as defined by the untranslated PCI address) does not complete before the DMA write from the previous DMA write is in the coherency domain.

R1–4.2.2.2–5. *(Requirement Number Reserved For Compatibility)*

R1–4.2.2.2–6. The hardware must make sure that all DMA write data buffered from an IOA, which is destined for system memory, is in the platform's coherency domain prior to delivering data from a *Load* operation through the same PHB which has come after the DMA write operation(s).

R1–4.2.2.2–7. The hardware must make sure that all DMA write data buffered from an IOA, which is destined for system memory, is in the platform's coherency domain prior to delivering an MSI from that same IOA which has come after the DMA write operation(s).

Architecture Notes:

- Requirement R1–4.2.2.2–4 clarifies (and may tighten up) the PCI architecture requirement that the read be to the "just-written" data.
- The address comparison for determining whether the address of the data being read is the same as the address of that being written is in the same cache line is based on the PCI address and not a TCE-translated address. This says that the System Memory cache line address will be the same also, since the requirement is directed towards operations under the same PHB. However, use of a DMA Read Request and DMA Write Request that use different PCI addresses (even if they hit the same System Memory address) are not required to be kept in order (see Requirement R1–4.2.8.1–1). So, for example, Requirement R1–4.2.2.2–4 says that split PHBs that share the same data buffers at the system end do not have to keep DMA Read Request following a DMA Write Request in order when they do not traverse the same PHB PCI bus (even if they get translated to the same system address) or when they originate on the same PCI bus but have different PCI bus addresses (even if they get translated to the same system address).
- Requirement R1–4.2.2.2–6 is the only case where the *Load* and *Store* paths are coupled to the DMA data path. This requirement guarantees that the software has a method for forcing DMA write data out of any buffers in

the path during the servicing of a completion interrupt from the IOA. Note that the IOA can perform the flush prior to the completion interrupt, via Requirement R1–4.2.2.2–4. That is, the IOA can issue a read request to the last word written and wait for the read completion data to return. When the read is complete, the data will have arrived at the destination. In addition, the use of MSIs, instead of LSIs, allows for a programming model for IOAs where the interrupt signalling itself pushes the last DMA write to System Memory, prior to the signalling of the interrupt to the system (see Requirement R1–4.2.2.2–7).

- A DMA read operation is allowed to be processed prior to the completion of a previous DMA read operation, but is not required to be.

4.2.2.3 PCI DMA Operations and Coherence

The I/O is not aware of the setting of the coherence required bit when performing operations to System Memory, and so the PHB needs to assume that the coherency is required.

R1–4.2.2.3–1. I/O transactions to System Memory through a PHB must be made with coherency required.

4.2.3 Byte Ordering Conventions

LoPAPR platforms operate with either Big-Endian (BE) or Little-Endian addressing. In Big-Endian systems, the address of a word in memory is the address of the most significant byte (the “big” end) of the word. Increasing memory addresses will approach the least significant byte of the word. In Little-Endian (LE) addressing, the address of a word in memory is the address of the least significant byte (the “little” end) of the word. See also Section 2.7, “Endian Support,” on page 53.

The PCI bus itself can be thought of as not inherently having an endianess associated with it (although its numbering convention indicates LE). It is the IOAs on the PCI bus that can be thought of as having endianess associated with them. Some PCI IOAs will contain a mode bit to allow them to appear as either a BE or LE IOA. Some IOAs will even have multiple mode bits; one for each data path (*Load* and *Store* versus DMA). In addition, some IOAs may have multiple concurrent apertures, or address ranges, where the IOA can be accessed as a LE IOA in one aperture and as a BE IOA in another.

R1–4.2.3–1. When the processor is operating in the Big-Endian mode, the platform design must produce the results indicated in Table 10, “Big-Endian Mode Load and Store Programming Considerations,” on page 79 while issuing *Load* and *Store* operations to various entities with various endianess.

R1–4.2.3–2. When performing DMA operations through a PHB, the platform must not modify the data during the transfer process; the lowest addressed byte in System Memory being transferred to the lowest addressed byte on the PCI bus, the second byte in System Memory being transferred as the second byte on the PCI bus, and so on.

Table 10. Big-Endian Mode *Load* and *Store* Programming Considerations

Destination	Transfer Operation
BE scalar entity: For example, TCE or BE register in a PCI IOA	*Load* or *Store*
LE scalar entity: For example, LE register in a PCI IOA or PCI Configuration Registers	*Load* or *Store Reverse*

4.2.4 PCI Bus Protocols

This section details the items from the *PCI Local Bus Specification* [18], *PCI-X Protocol Addendum to the PCI Local Bus Specification* [21], and *PCI Express Base Specification* [22] documents where there is variability allowed, and therefore further specifications, requirements, or explanations are needed.

Specifically, Table 11, "PCI Express Optional Feature Usage in LoPAPR Platforms," on page 80 details specific PCI Express options and the requirements for usage of such in LoPAPR platforms. These requirements will drive the design of PHB implementations. See the *PCI Express Base Specification* [22] for more information.

Table 11. PCI Express Optional Feature Usage in LoPAPR Platforms

PCI Express Option Name	Usage		Description
	Base	IBM Server	
Usage Legend: NS = Not Supported; O = Optional (see also Description); OR = Optional but Recommended; R = Required; SD = See Description			
Peripheral I/O Space	SD	SD	Required if the platform is going to support any Legacy I/O devices, as defined by the *PCI Express Base Specification* [22], otherwise support not required. The expectation is that Legacy I/O device support by PHBs will end soon, so platform designers should not rely on this being there when choosing I/O devices.
64-bit DMA addresses	O	OR SD	Implementation is optional, but is expected to be needed in some platforms, especially those with more complex PCI Express fabrics. Although the **"ibm,dma-window"** property can implement 64-bit addresses, some OSs and Device Drivers may not be able to handle values in the **"ibm,dma-window"** property that are greater than or equal to 4 GB. Therefore, it is recommended that 64-bit DMA addresses be implemented through the Dynamic DMA Window option (see Section 7.4.10, "DMA Window Manipulation Calls," on page 259).
Advanced Error Reporting (AER)	O	R SD	This has implications in the IOAs selected for use in the platform, as well as the PHB and firmware implementation. See the *PCI Express Base Specification* [22].
PCIe Relaxed Ordering (RO) and ID-Based Ordering (IDO)	NS	NS	Enabling either of these options could allow DMA transactions that should be dropped by an EEH Stopped State, to get to the system before the EEH Stopped State is set, and therefore these options are not to be enabled. Specifically, either of these could allow DMA transactions that follow a DMA transaction in error to bypass the PCI Express error message signalling an error on a previous packet. **Platform Implementation Note:** It is permissible for the platform (for example, the PHB or the nest) to re-order DMA transactions that it knows can be re-ordered -- such as DMA transactions that come from different Requester IDs or come into different PHBs -- as long as the ordering with respect to error signalling is met.
5.0 GT/s signalling (Gen 2)	O	OR	
8 GT signalling (Gen 3)	O	OR	
TLP Processing Hints	O	O	If implemented, it must be transparent to OSs.
Atomic Operations (32 and 64 bit)	O	OR SD	May be required if the IOAs being supported require it. May specifically be needed for certain classes of IOAs such as accelerators.
Atomic Operations (128 bit)	O	OR SD	When 128 bit Atomic Operations are supported, 32 and 64 bit Atomic Operations must be also supported.
Resizable BAR	O	O	
Dynamic Power Allocation (DPA)	NS	NS	No support currently defined in LoPAPR.
Latency Tolerance Reporting (LTR)	NS	NS	No support currently defined in LoPAPR.

Table 11. PCI Express Optional Feature Usage in LoPAPR Platforms ***(Continued)***

PCI Express Option Name	Usage		Description
	Base	IBM Server	
Usage Legend: NS = Not Supported; O = Optional (see also Description); OR = Optional but Recommended; R = Required; SD = See Description			
Optimized Buffer Flush/Fill (OBFF)	NS	NS	No support currently defined in LoPAPR.
PCIe Multicast	NS	NS	No support currently defined in LoPAPR.
Alternative Routing ID Interpretation (ARI)	O	SD	Required when the platform will support PCI Express IOV IOAs.
Access Control Services (ACS)	SD	SD	It is required that peer to peer operation between IOAs be blocked when LPAR is implemented and those IOAs are assigned to different LPAR partitions. For switches below a PHB, when the IOA functions below the switch may be assigned to different partitions, this blocking is provided by ACS in the switch. This is required even in Base platforms, if the above conditions apply.
Function Level Reset (FLR)	SD	SD	Required when a PE consists of something other than a full IOA. For example, if each function of a multi-function IOA each is in its own PE. An SR-IOV Virtual Function (VF) may be one such example.
End-to-End CRC (ECRC)	O	R SD	This has implications in the IOAs selected for use in the platform, as well as the PHB and firmware implementation. See the *PCI Express Base Specification* [22].
Internal Error Reporting	OR SD	OR SD	Implement where appropriate. Platforms need to consider this for platform switches, also. PHBs may report internal errors to firmware using a different mechanism outside of this architecture.
Address Translation Services (ATS)	NS	NS	LoPAPR does not support ATS, because the invalidation and modification of the Address Translation and Protection Table (ATPT) -- called the TCEs in LoPAPR -- is a synchronous operations, whereas the ATS invalidation requires a more asynchronous operation.
Page Request Interface (PRI)	NS	NS	Requires ATS, which is not supported by LoPAPR.
Single Root I/O Virtualization (SR-IOV)	O	OR	It is likely that most server platforms will need to be enabled to use SR-IOV IOAs.
Multi-Root I/O Virtualization (MR-IOV)	SD	SD	Depending on how this is implemented, an MR-IOV device is likely to look like an SR-IOV device to an OS (with the platform hiding the Multi-root aspects). PHBs may be MR enabled or the MR support may be through switches external to the PHBs.

4.2.5 Programming Model

Normal memory mapped *Load* and *Store* instructions are used to access a PHB's facilities or PCI IOAs on the I/O side of the PHB. Chapter 3, "Address Map," on page 59 defines the addressing model. Addresses of IOAs are passed by OF via the OF device tree.

R1–4.2.5–1. If a PHB defines any registers that are outside of the PCI Configuration space, then the address of those registers must be in the Peripheral Memory Space or Peripheral I/O Space for that PHB, or must be in the System Control Area.

PCI master DMA transfers refer to data transfers between a PCI master IOA and another PCI IOA, or System Memory, where the PCI master IOA supplies the addresses and controls all aspects of the data transfer. Transfers from a PCI master to the PCI I/O Space are essentially ignored by a PHB (except for address parity checking). Transfers from a PCI master to PCI Memory Space are either directed at PCI Memory Space (for peer to peer operations) or need to be directed to the host side of the PHB. DMA transfers directed to the host side of a PHB may be to System Memory or may be to another IOA via the Peripheral Memory Space of another HB. Transfers that are directed to the Peripheral I/O Space of another HB are considered to be an addressing error (see Chapter 10, "Error and Event Notification," on

page 281). For information about decoding these address spaces and the address transforms necessary, see Chapter 3, “Address Map,” on page 59.

4.2.6 Peer-to-Peer Across Multiple PHBs

This architecture does not architect peer-to-peer traffic between two PCI IOAs when the operation traverses multiple PHBs.

R1–4.2.6–1. The platform must prevent Peer-to-Peer operations that would cross multiple PHBs.

4.2.7 Dynamic Reconfiguration of I/O

Disconnecting or connecting an I/O subsystem while the system is operational and then having the new configuration be operational, including any new added subsystems, is a subset of Dynamic Reconfiguration (DR).

Some platforms may also support plugging/unplugging of PCI IOAs while the system is operational. This is another subset of DR.

DR is an option and as such, is not required by this architecture. Attempts to change the hardware configuration on a platform that does not enable configuration change, whose OS does not support that configuration change, or without the appropriate user configuration change actions may produce unpredictable results (for example, the system may crash).

PHBs in platforms that support the PCI Hot Plug Dynamic Reconfiguration (DR) option may have some unique design considerations. For information about the DR options, see Chapter 13, “Dynamic Reconfiguration (DR) Architecture,” on page 355.

4.2.8 Split Bridge Implementations

In some platforms the PHB may be split into two pieces, separated by a cable or fiber optics. The piece that is connected to the system bus (or switch) and which generates the interconnect is called the Hub. There are several implications of such implementations and several requirements to go along with these.

4.2.8.1 Coherency Considerations with IOA to IOA Communications via System Memory

Bridges which are split across multiple chips may introduce a large enough latency between the time DMA write data is accepted by the PHB and the time that previously cached copies of the same System Memory locations are invalidated, and this latency needs to be taken into consideration in designs, as it can introduce the problems described below. This is not a problem if the same PCI address is used under a single PHB by the same or multiple IOAs, but can be a problem under any of the following conditions:

- The same PCI address is used by different IOAs under different PHBs.
- Different PCI addresses are used which access the same System Memory coherency block, regardless of whether the IOA(s) are under the same PHB or not; for example:
  - Two different TCEs accessing the same System Memory coherency block.

An example scenario where this could be a problem is as follows:

1. Device 1 does a DMA read from System Memory address x using PCI address y
2. Device 2 (under same PHB as Device 1 -- the devices could even be different function in the same IOA) does a DMA write to System Memory address x using PCI address z.

3. Device 2 attempts to read back System Memory address x before the time that its previous DMA write is globally coherent (that is, before the DMA write gets to the Hub and an invalidate operation on the cache line containing that data gets back down to the PHB), and gets the data read by Device 1 rather than what it just wrote.

Another example scenario is as follows:

1. Device 1 under PHB 1 does a DMA read from System Memory location x.
2. Device 2 under PHB 2 does a DMA write to System Memory location x and signals an interrupt to the system.
3. The interrupt bypasses the written data which is on its way to the coherency domain.
4. The device driver for Device 2 services the interrupt and signals Device 1 via a *Store* to Device 1 that the data is there at location x.
5. Device 1 sees the *Store* before the invalidate operation on the cache line containing the data propagates down to invalidate the previous cached copy of x, and does a DMA read of location x using the same address as in step (1), getting the old copy of x instead of the new copy.

This last example is a little far-fetched since the propagation times should not be longer than the interrupt service latency time, but it is possible. In this example, the device driver should do a *Load* to Device 2 during the servicing of the interrupt and wait for the *Load* results before trying to signal Device 1, just the way that this device driver would to a *Load* if it was a program which was going to use the data written instead of another IOA. Note that this scenario can also be avoided if the IOA uses a PCI Message Signalled Interrupt (MSI) rather than the PCI interrupt signals pins, in order to signal the interrupt (in which case the *Load* operation is avoided).

R1–4.2.8.1–1. A DMA read to a PCI address which is different than a PCI address used by a previous DMA write or which is performed under a different PHB must not presume that a previous DMA write is complete, even if the DMA write is to the same System Memory address, unless one of the following is true:

- The IOA doing the DMA write has followed that write by a DMA read to the address of the last byte of DMA write data to be flushed (the DMA read request must encompass the address of the last byte written, but does not need to be limited to just that byte) and has waited for the results to come back before an IOA is signaled (via peer-to-peer operations or via software) to perform a DMA read to the same System Memory address.
- The device driver for the IOA doing the DMA write has followed that write by a *Load* to that IOA and has waited for the results to come back before a DMA read to the same System Memory address with a different PCI address is attempted.
- The IOA doing the DMA write has followed the write with a PCI Message Signalled Interrupt (MSI) as a way to interrupt the device driver, and the MSI message has been received by the interrupt controller.

4.3 I/O Bus to I/O Bus Bridges

The PCI bus architecture was designed to allow for bridging to other slower speed I/O buses or to another PCI bus. The requirements when bridging from one I/O bus to another I/O bus in the platform are defined below.

R1–4.3–1. All bridges must comply with the bus specification(s) of the buses to which they are attached.

4.3.1 What Must Talk to What

Platforms are not required to support peer to peer operations between IOAs. IOAs on the same shared bus segment will generally be able to do peer to peer operations between themselves. Peer to peer operations in an LPAR environment, when the operations are between IOAs that are not in the same partition, is specifically prohibited (see Requirement R1–14.3–4).

4.3.2 PCI to PCI Bridges

This architecture allows the use of PCI to PCI bridges and PCI Express switches in the platform. TCEs are used with the IOAs attached to the other side of the PCI to PCI bridge or PCI Express switch when those IOAs are accessing something on the processor side of the PHB. After configuration, PCI to PCI bridges and PCI Express switches are basically transparent to the software as far as addressing is concerned (the exception is error handling). For more information, see the appropriate PCI Express switch specification.

R1–4.3.2–1. Conventional PCI to PCI bridges used on the base platform and plug-in cards must be compliant with the most recent version of the *PCI-to-PCI Bridge Architecture Specification* [19] at the time of the platform design, including any approved Engineering Change Requests (ECRs) against that document. PCI-X to PCI-X bridges used on the base platform and plug-in cards must be compliant with the most recent version of the *PCI-X Protocol Addendum to the PCI Local Bus Specification* [21] at the time of the platform design, including any approved Engineering Change Requests (ECRs) against that document.

R1–4.3.2–2. PCI Express to PCI/PCI-X and PCI/PCI-X to PCI Express bridges used on the base platform and plug-in cards must be compliant with the most recent version of the *PCI Express to PCI/PCI-X Bridge Specification* [23] at the time of the platform design, including any approved Engineering Change Requests (ECRs) against that document.

R1–4.3.2–3. PCI Express switches used on the base platform and plug-in cards must be compliant with the most recent version of the *PCI Express Base Specification* [22] at the time of the platform design, including any approved Engineering Change Requests (ECRs) against that document.

R1–4.3.2–4. Bridges and switches used in platforms which will support PCI Express IOAs beneath them must support pass-through of PCI configuration cycles which access the PCI extended configuration space.

Software and Platform Implementation Notes:

1. Bridges used on plug-in cards that do not follow Requirement R1–4.3.2–4 will presumably allow for the operation of their IOAs on the plug-in card, even though not supporting the PCI extended configuration address space, because the card was designed with the bridges and IOAs in mind.

2. Determination of support of the PCI configuration address space is via the **`"ibm,pci-config-space-type"`** property in the IOA's node.

R1–4.3.2–5. Bridges and switches used in platforms which will support PCI Express IOAs beneath them must support 64-bit addressing.

4.4 Bridge Extensions

4.4.1 Enhanced I/O Error Handling (EEH) Option

The EEH option uses the following terminology.

PE	A Partitionable Endpoint. This refers to the granule that is treated as one for purposes of EEH recovery and for assignment to an OS image (for example, in an LPAR environment). Note that the PE granularity supported by the hardware may be finer than is supported by the firmware. See also Section 4.1, "I/O Topologies and Endpoint Partitioning," on page 71. A PE may be any one of the following: A single-function or multi-function IOA A set of IOAs and some piece of I/O fabric above the IOAs that consists of one or more bridges or switches.
EEH Stopped state	The state of a PE being in both the MMIO Stopped state and DMA Stopped state.
MMIO Stopped state	The state of the PE which will discard any MMIO *Store*s to that PE, and will return all-1's data for *Load*s to that PE. If the PE is in the MMIO Stopped state and EEH is disabled, then a *Load* will also return a machine check to the processor that issued the *Load*, for the *Load* that had the initial error and while the PE remains in the MMIO Stopped state.
DMA Stopped state	The state of the PE which will block any further DMA requests from that PE (DMA completions that occur after the DMA Stopped state is entered that correspond to DMA requests that occurred before the DMA Stopped state is entered, may be completed).
Failure	A detected error between the PE and the system (for example, processor or memory); errors internal to the PE are not considered failures unless the PE signals the error via a normal I/O fabric error signalling protocol. (for example, SERR or ERR_FATAL).

The Enhanced I/O Error Handling (EEH) option is defined primarily to enhance the system recoverability from failures that occur during *Load* and *Store* operations. In addition, certain failures that are normally non-recoverable during DMA are prevented from causing a catastrophic failure to the system (for example, a conventional PCI address parity error).

The basic concept behind the EEH option is to turn all failures that cannot be reported to the IOA, into something that looks like a conventional PCI Master Abort (MA) error[1] on a *Load* or *Store* operation to the PE during and after the failure; responding with all-1's data and no error indication on a *Load* instruction and ignoring *Store* instructions. The MA error should be handled by a device driver, so this approach should just be an extension to what should be the error handling without this option implemented.

The following is the general idea behind the EEH option:

- On a failure that occurs in an operation between the PHB and PE:
  - Put the PE into the MMIO Stopped and DMA Stopped states (also known as the EEH Stopped state). This is defined as a state where the PE is prevented from doing any further operations that could corrupt the system; which for the most part means blocking DMA from the PE and preventing load and store completions to the PE.

1.A conventional PCI MA error is where the conventional PCI IOA does not respond as a target with a device select indication (that is, the IOA does not respond by activating the DEVSEL signal back to the master). For PCI Express, the corresponding error is Unsupported Request (UR).

- While the PE is in the MMIO Stopped state, if a *Load* or *Store* is targeted to that PE, then return all-1's data with no error indication on a *Load* and discard all *Stores* to that PE. That is, essentially treat the *Load* or *Store* the same way as if a MA error was received on that operation.
- The device driver and OS recovers a PE by removing it from the MMIO Stopped state (keeping it in the DMA Stopped state) and doing any necessary loads to the PE to capture PE state, and then either doing the necessary stores to the PE to set the appropriate state before removing the PE from the DMA Stopped state and continuing operations, or doing a reset of the PE and then re-initializing and restarting the PE.[1]

- In order to make sure that there are no interactions necessary between device drivers during recovery operations, each PE will have the capability of being removed from its MMIO Stopped and DMA Stopped states independent from any other PE which is in the MMIO Stopped or DMA Stopped state.
- In order to take into account device drivers which do not correctly implement MA recovery, make sure that the EEH option can be enabled and disabled independently for each PE.[2]
- EEH, as defined, only extends to operations between the processor and a PE and between a PE and System Memory. It does not extend to direct IOA to IOA peer to peer operations.

Hardware changes for this option are detailed in the next section. RTAS changes required are detailed in Section 7.3.11, "Enhanced I/O Error Handling (EEH) Option Functions," on page 176.

4.4.1.1 EEH Option Requirements

Although the EEH option architecture may be extended to other I/O topologies in the future, for now this recovery architecture will be limited to PCI.

In order to be able to test device driver additional code for the EEH-enabled case, the EEH option also requires the Error Injection option be implemented concurrently.

The additional requirements on the hardware for this option are as follows. For the RTAS requirements for this option, see Section 7.3.11, "Enhanced I/O Error Handling (EEH) Option Functions," on page 176.

R1–4.4.1.1–1. For the EEH option: A platform must implement the Error Injection option concurrently with the EEH option, with an error injection granularity to the PE level.

R1–4.4.1.1–2. For the EEH option: If a platform is going to implement the EEH option, then the I/O topology implementing EEH must only consist of PCI components.

R1–4.4.1.1–3. For the EEH option: The hardware must provide a way to independently enable and disable the EEH option for each PE with normal processor *Load* and *Store* instructions, and must provide the capability of doing this while not disturbing operations to other PEs in the platform.

R1–4.4.1.1–4. For the EEH option: The hardware fault isolation register bits must be set the same way on errors when the EEH option is enabled as they were when the EEH option is not implemented or when it is implemented but disabled.

R1–4.4.1.1–5. For the EEH option: Any detected failure to/from a PE must set both the MMIO Stopped and DMA Stopped states for the PE, unless the error that caused the failure can be reported to the IOA in a way that the IOA will report the error to its device driver in a way that will avoid any data corruption.

1.Most device drivers will implement a reset and restart in order to assure a clean restart of operations.

2.LPAR implementations limit the capability of running with EEH disabled (see Requirement R1–14.3–1 and Requirement R1–14.3–2).

R1–4.4.1.1–6. For the EEH option: If an I/O fabric consists of a hierarchy of components, then when a failure is detected in the fabric, all PEs that are downstream of the failure must enter the MMIO Stopped and DMA Stopped states if they may be affected by the failure.

R1–4.4.1.1–7. For the EEH option: While a PE has its EEH option enabled, if a failure occurs, the platform must not propagate it to the system as any type of error (for example, as an SERR for a PE which is a conventional PCI-to-PCI bridge).

R1–4.4.1.1–8. For the EEH option: From the time that the MMIO Stopped state is entered for a PE, the PE must be prevented from responding to *Load* and *Store* operations including the operation that caused the PE to enter the MMIO Stopped state; a *Load* operation must return all-1's with no error indication and a *Store* operation must be discarded (that is, *Load* and *Store* operations being treated like they received a conventional PCI Master Abort error), until one of the following is true:

a. The *ibm,set-eeh-option* RTAS call is called with *function* 2 (Release PE for MMIO *Load*/*Store* operations).

b. The *ibm, set-slot-reset* RTAS call is called with *function* 0 (Deactivate the reset signal to the PE).

c. The power is cycled (off then on) to the PE.

d. The partition or system is rebooted.

R1–4.4.1.1–9. For the EEH option: From the time that the DMA Stopped state is entered for a PE, the PE must be prevented from initiating a new DMA request or completing a DMA request that caused the PE to enter the DMA Stopped state (DMA requests that were started before the DMA Stopped State is entered may be completed), and including MSI DMA operations, until one of the following is true:

a. The *ibm,set-eeh-option* RTAS call is called with *function* 3 (Release PE for DMA operations).

b. The *ibm, set-slot-reset* RTAS call is called with *function* 0 (Deactivate the reset signal to the PE).

c. The power is cycled (off then on) to the PE.

d. The partition or system is rebooted.

R1–4.4.1.1–10. For the EEH option: The hardware must provide the capability to the firmware to determine, on a per-PE basis, that a failure has occurred which has caused the PE to be put into the MMIO Stopped and DMA Stopped states and to read the actual state information (MMIO Stopped state and DMA Stopped state).

R1–4.4.1.1–11. For the EEH option: The hardware must provide the capability of separately enabling and resetting the DMA Stopped and MMIO Stopped states for a PE without disturbing other PEs on the platform. The hardware must provide this capability without requiring a PE reset and must do so through normal processor *Store* instructions.

R1–4.4.1.1–12. For the EEH option: The hardware must provide the capability to the firmware to deactivate the reset to each PE, independent of other PEs, and the hardware must provide the proper controls on the reset transitions in order to prevent failures from being introduced into the platform by the changing of the reset.

R1–4.4.1.1–13. For the EEH option: The hardware must provide the capability to the firmware to activate the reset to each PE, independent of other PEs, and the hardware must provide the proper controls on the reset transitions in order to prevent failures from being introduced into the platform by the changing of the reset.

R1–4.4.1.1–14. For the EEH option: The hardware must provide the capability to the firmware to read the state of the reset signal to each PE.

R1–4.4.1.1–15. For the EEH option: When a PE is put into the MMIO Stopped and DMA Stopped states, it must be done in such a way to not introduce failures that may corrupt other parts of the platform.

R1–4.4.1.1–16. For the EEH option: The hardware must allow firmware access to internal bridge and I/O fabric control registers when any or all of the PEs are in the MMIO Stopped state.

Platform Implementation Note: It is expected that bridge and fabric control registers will have their own PE state separate from the PEs for IOAs.

R1–4.4.1.1–17. For the EEH option: A PE that supports the EEH option must not share an interrupt with another PE in the platform.

Hardware Implementation Notes:

1. Requirement R1–4.4.1.1–4 means that the hardware must always update the standard PCI error/status registers in the bus' configuration space as defined by the bus architecture, even when the EEH option is enabled.

2. The type of error information trapped by the hardware when a PE is placed into the MMIO Stopped and DMA Stopped states is implementation dependent. It is expected that the system software will do an *check-exception or ibm,slot-error-detail* RTAS call to gather the error information when a failure is detected.

3. A DMA operation (Read or Write) that was initiated before a *Load*, *Store*, or DMA error, does not necessarily need to be blocked, as it was not a result of the *Load*, *Store*, or DMA that failed. The normal PCI Express ordering rules require that an ERR_FATAL or ERR_NONFATAL from a failed *Store* or DMA error, or a *Load* Completion with error status, will reach the PHB prior to any DMA that might have been kicked-off in error as a result of a failed *Load* or *Store* or a *Load* or *Store* that follows a failed *Load* or *Store*. This means that as long as the PHB processes an ERR_FATAL, ERR_NONFATAL, or *Load* Completion which indicates a failure, prior to processing any more DMA operations or *Load* Completions, and puts the PE into the MMIO and Stopped DMA Stopped states, implementations should be able to block DMA operations that were kicked-off after a failing DMA operation and allow DMA operations that were kicked off before a failing DMA operation without violating the normal PCI Express ordering rules.

4. In reference to Requirements R1–4.4.1.1–5, and R1–4.4.1.1–6, PCI Express implementations may choose to enter the MMIO Stopped and DMA Stopped states even if an error can be reported back to the IOA.

R1–4.4.1.1–18. For the EEH option: If the device driver(s) for any IOA(s) in a PE in the platform are EEH unaware (that is may produce data integrity exposures due to a MMIO Stopped or DMA Stopped state), then the firmware must prevent the IOA(s) in such a PE from being enabled for operations (that is, do not allow the Bus Master, Memory Space or I/O Space bits in the PCI configuration Command register from being set to a 1) while EEH is enabled for that PE, and instead of preventing the PE from being enabled, may instead turn off EEH when such an enable is attempted without first an attempt by the device driver to enable EEH (by the *ibm,set-eeh-option*), providing such EEH disablement does not violate any other requirement for EEH enablement (for example, Requirement R1–14.3–1 or R1–14.3–2).

Software Implementation Note: To be EEH aware, a device driver does not need to be able to recover from an MMIO Stopped or DMA Stopped state, only recognize the all-1's condition and not use data from operations that may have occurred since the last all-1's checkpoint. In addition, the device driver under such failure circumstances needs to turn off interrupts (using the *ibm,set-int-off* RTAS call or by resetting the PE and keeping it reset with *ibm,set-slot-reset* or *ibm,slot-error-detail*) in order to make sure that any (unserviceable) interrupts from the PE do not affect the system. Note that this is the same device driver support needed to protect against an IOA dying or against a no-DEVSEL type error (which may or may not be the result of an IOA that has died).

4.4.1.2 Slot Level EEH Event Interrupt Option

Some platform implementations may allow asynchronous notification of EEH events via an external interrupt. This is called the Slot Level EEH Event Interrupt option. When implemented, the platform will implement the **`"ibm,io-events-capable"`** property in the nodes where the EEH control resides, and the *ibm,set-eeh-option* RTAS call will implement function 4 to enable the EEH interrupt for each of these nodes and function 5 to disable the EEH interrupt for each of these nodes (individual control by node). Calling the *ibm,set-eeh-option* RTAS call with

function 4 or function 5 when the node specified does not implement this capability will return a -3, indicating invalid parameters.

The interrupt source specified in the **ibm,io-events** child must be enabled (in addition to any individual node enables) via the *ibm,int-on* RTAS call and the priority for that interrupt, as set in the XIVE by the *ibm,set-xive* RTAS call, must be something other than 0xFF, in order for the external interrupt to be presented to the system.

The **"ibm,io-events-capable"** property, when it exists, contains 0 to N interrupt specifiers (per the definition of interrupt specifiers for the node's interrupt parent). When no interrupt specifiers are specified by the **"ibm,io-events-capable"** property, then the interrupt, if enabled, is signaled via the interrupt specifier given in the **ibm,io-events** child node of the **/events** node.

R1–4.4.1.2–1. For the Slot Level EEH Event Interrupt option: All of the following must be true:

a. The platform must implement the **"ibm,io-events-capable"** property in all device tree nodes which represent bridge where EEH is implemented and for which the EEH io-event interrupt is to be signaled.

b. The platform must implement functions 4 and 5 of the *ibm,set-eeh-option* RTAS call for all PEs under nodes that contain the **"ibm,io-events-capable"** property.

4.4.2 Error Injection (ERRINJCT) Option

The Error Injection (ERRINJCT) option is defined primarily to test enhanced error recovery software. As implemented in the I/O bridge, this option is used to test the software which implements the recovery which is enabled by the EEH option in that bridge. Specifically, the *ioa-bus-error* and *ioa-bus-error-64* functions of the *ibm,errinjct* RTAS call are used to inject errors onto each PE primary bus, which in turn will cause certain actions on the bus and certain actions by the PE, the EEH logic, and by the error recovery software.

4.4.2.1 ERRINJCT Option Hardware Requirements

Although the *ioa-bus-error* and *ioa-bus-error-64* functions of the *ibm,errinjct* RTAS call may be extended to other I/O buses and PEs in the future, for now this architecture will be limited to PCI buses.

The type of errors, and the injection qualifiers, place the following additional requirements on the hardware for this option.

R1–4.4.2.1–1. For the *ioa-bus-error* and *ioa-bus-error-64* functions of the Error Injection option: If a platform is going to implement either of these functions of this option, then the I/O topology must be PCI.

R1–4.4.2.1–2. For the *ioa-bus-error* and *ioa-bus-error-64* functions of the Error Injection option: The hardware must provide a way to inject the required errors for each PE primary bus, and the errors must be injectable independently, without affecting the operations on the other buses in the platform.

R1–4.4.2.1–3. For the *ioa-bus-error* and *ioa-bus-error-64* functions of the Error Injection option: The hardware must provide a way to set up for the injection of the required errors without disturbing operations to other buses outside the PE.

R1–4.4.2.1–4. For the *ioa-bus-error* and *ioa-bus-error-64* functions of the Error Injection option: The hardware must provide a way to the firmware to set up the following information for the error injection operation by normal processor *Load* and *Store* instructions:

- Address at which to inject the error
- Address mask to mask off any combination of the least significant 24 (64 for the *ioa-bus-error-64* function) bits of the address

- PE primary bus number which is to receive the error
- Type of error to be injected

R1–4.4.2.1–5. For the *ioa-bus-error* and *ioa-bus-error-64* functions of the Error Injection option: The platform must have the capability of selecting the errors specified in Table 12, “Supported Errors for Conventional PCI, PCI-X Mode 1 or PCI-X Mode 2 Error Injectors,” on page 90 when the bus directly below the bridge injecting the error is a Conventional PCI or PCI-X Bus, and the errors specified in Table 13, “Supported Errors for PCI Express Error Injectors,” on page 91 when the bus directly below the bridge injecting the error is a PCI Express link, and when that error is appropriate for the platform configuration, and the platform must limit the injection of errors which are inappropriate for the given platform configuration.

Platform Implementation Note: As an example of inappropriate errors to inject in Requirement R1–4.4.2.1–5, consider the configuration where there is an I/O bridge or switch below the bridge with the injector and that bridge generates multiple PEs and when those PEs are assigned to different LPAR partitions. In that case, injection of some real errors may cause the switches or bridges to react and generate an error that affects multiple partitions, which would be inappropriate. Therefore, to comply with Requirement R1–4.4.2.1–5, the platform may either emulate some errors in some configurations instead of injecting real errors on the link or bus, or else the platform may not support injection at all to those PEs. Another example where a particular error may be inappropriate is when there is a heterogeneous network between the PHB and the PE (for example, a PCI Express bridge that converts from a PCI Express PHB and a PCI-X PE).

Table 12. Supported Errors for Conventional PCI, PCI-X Mode 1 or PCI-X Mode 2 Error Injectors

Operation	PCI Address Space(s)	Error (s)	Other Requirements
Load	Memory, I/O, Config	Data Parity Error	All PCI-X adapters operating in Mode 2 and some operating in Mode 1 utilize a double bit detecting, single bit correcting Error Correction Code (ECC). In these cases, ensure that at least two bits are modified to detect this error.
		Address Parity Error	
Store	Memory, I/O, Config	Data Parity Error	
		Address Parity Error	
DMA read	Memory	Data Parity Error	All PCI-X adapters operating in Mode 2 and some operating in Mode 1 utilize a double bit detecting, single bit correcting Error Correction Code (ECC). In these cases, ensure that at least two bits are modified to detect this error.
		Address Parity Error	
		Master Abort	
		Target Abort	
DMA write	Memory	Data Parity Error	All PCI-X adapters operating in Mode 2 and some operating in Mode 1 utilize a double bit detecting, single bit correcting Error Correction Code (ECC). In these cases, ensure that at least two bits are modified to detect this error.
		Address Parity Error	
		Master Abort	
		Target Abort	

Table 13. Supported Errors for PCI Express Error Injectors

Operation	PCI Address Space(s)	Error (s)	Other Requirements
Load	Memory, I/O, Config	TLP ECRC Error	The TLP ECRC covers the address and data bits of a TLP. Therefore, one cannot determine if the integrity error resides in the address or data portion of a TLP.
Store	Memory, I/O, Config	TLP ECRC Error	
DMA read	Memory	TLP ECRC Error	The TLP ECRC covers the address and data bits of a TLP. Therefore, one cannot determine if the integrity error resides in the address or data portion of a TLP.
		Completer Abort or Unsupported Request	Inject the error that is injected on a TCE Page Fault.
DMA write	Memory	TLP ECRC Error	The TLP ECRC covers the address and data bits of a TLP. Therefore, one cannot determine if the integrity error resides in the address or data portion of a TLP.

R1–4.4.2.1–6. For the *ioa-bus-error* and *ioa-bus-error-64* functions of the Error Injection option: The hardware must provide a way to inject the errors in Table 13, “Supported Errors for PCI Express Error Injectors,” on page 91 in a non-persistent manner (that is, at most one injection for each invocation of the *ibm,errinjct* RTAS call).

4.4.2.2 ERRINJCT Option OF Requirements

The Error Injection option will be disabled for all IOAs prior to the OS getting control.

R1–4.4.2.2–1. For the *ioa-bus-error* and *ioa-bus-error-64* functions of the Error Injection option: The OF must disable the ERRINJCT option for all PEs and all empty slots on all bridges which implement this option prior to passing control to the OS.

Hardware and Firmware Implementation Note: The platform only needs the capability to setup the injection of one error at a time, and therefore injection facilities can be shared. The *ibm,open-errinjct* and *ibm,close-errinjct* are used to make sure that only one user is using the injection facilities at a time.

4.4.3 Bridged-I/O EEH Support Option

If a platform requires multi-function I/O cards which are constructed by placing multiple IOAs beneath a PCI to PCI bridge, then extra support is needed to support such cards in an EEH-enabled environment. If this option is implemented, then the *ibm,configure-bridge* RTAS call will be implemented and therefore the **`“ibm,configure-bridge”`** property will exist in the **`rtas`** device node.

R1–4.4.3–1. For the Bridged-I/O EEH Support option: The platform must support the *ibm,configure-bridge* RTAS call.

R1–4.4.3–2. For the Bridged-I/O EEH Support option: The OS must provide the correct EEH coordination between device drivers that control multiple IOAs that are in the same PE.

5 Processor and Memory

The purpose of this chapter is to specify the processor and memory requirements of this architecture. The processor architecture section addresses differences between the processors in the PA family as well as their interface variations and features of note. The memory architecture section addresses coherency, minimum system memory requirements, memory controller requirements, and cache requirements.

5.1 Processor Architecture

The Processor Architecture (PA) governs software compatibility at an instruction set and environment level. However, each processor implementation has unique characteristics which are described in its user's manual. To facilitate shrink-wrapped software, this architecture places some limitations on the variability in processor implementations. Nonetheless, evolution of the PA and implementations creates a need for both software and hardware developers to stay current with its progress. The following material highlights areas deserving special attention and provides pointers to the latest information.

5.1.1 Processor Architecture Compliance

The PA is defined in *Power ISA* [1].

R1–5.1.1–1. Platforms must incorporate only processors which comply fully with *Power ISA* [1].

R1–5.1.1–2. For the Symmetric Multiprocessor option: Multiprocessing platforms must use only processors which implement the processor identification register.

R1–5.1.1–3. Platforms must incorporate only processors which implement *tlbie* and *tlbsync*, and *slbie* and *slbia* for 64-bit implementations.

R1–5.1.1–4. Except where specifically noted otherwise in Section 5.1.4, "PA Features Deserving Comment," on page 95, platforms must support all functions specified by the PA.

Hardware and Software Implementation Note: The PA and this architecture view *tlbia* as an optional performance enhancement. Processors need not implement *tlbia*. Software that needs to purge the TLB should provide a sequence of instructions that is functionally equivalent to *tlbia* and use the content of the OF device tree to choose the software implementation or the hardware instruction. See Section 5.1.2, "PA Processor Differences," on page 93 for details.

5.1.2 PA Processor Differences

A complete understanding of processor differences may be obtained by studying *Power ISA* [1] and the user's manuals for the various processors.

The creators of this architecture cooperate with processor designers to maintain a list of supported differences, to be used by the OS instead of the processor version number (PVN), enabling execution on future processors. OF communicates these differences via properties of the **`cpu`** node of the OF device tree. Examples of OF device tree properties

which support these differences include **"64-bit"** and **"performance-monitor"**. See Appendix C, "PA Processor Binding," on page 753 for a complete listing and more details.

R1–5.1.2–1. The OS must use the properties of the **cpu** node of the OF device tree to determine the programming model of the processor implementation.

R1–5.1.2–2. The OS must provide an execution path which uses the properties of the **cpu** node of the OF device. The PVN is available to the platform aware OS for exceptional cases such as performance optimization and errata handling.

R1–5.1.2–3. The OS must support the 64-bit page table formats defined by *Power ISA* [1].

R1–5.1.2–4. Processors which exhibit the **"64-bit"** property of the **cpu** node of the OF device tree must also implement the "bridge architecture," an option in *Power ISA* [1].

R1–5.1.2–5. Platforms must restrict their choice of processors to those whose programming models may be described by the properties defined for the **cpu** node of the OF device tree in Appendix C, "PA Processor Binding," on page 753.

R1–5.1.2–6. Platform firmware must initialize the second and third pages above *Base* correctly for the processor in the platform prior to giving control to the OS.

R1–5.1.2–7. OS and application software must not alter the state of the second and third pages above *Base*.

R1–5.1.2–8. Platforms must implement the **"ibm,platform-hardware-notification"** property (see Appendix B, "LoPAPR Binding," on page 661) and include all PVRs that the platform may contain.

5.1.2.1 64-bit Implementations

Some 64-bit processor implementations will not support the full virtual address allowed by *Power ISA* [1]. As a result, this architecture adds a 64-bit virtual address subset to the PA and the corresponding **cpu** node property **"64-bit-virtual-address"** to OF.

In order for an OS to make use of the increased addressability of 64-bit processor implementations:

- The memory subsystem must support the addressing of memory located at or beyond 4 GB, and
- Any system memory located at or beyond 4 GB must be reported via the OF device tree.

At an abstract level, the effort to support 64-bit architecture in platforms is modest. The requirements follow.

R1–5.1.2.1–1. The OS must support the 64-bit virtual address subset, but may defer support of the full 80-bit virtual address until such time as it is required.

R1–5.1.2.1–2. Firmware must report the **"64-bit-virtual-address"** property for processors which implement the 64-bit virtual address subset.

R1–5.1.2.1–3. RTAS must be capable of being instantiated in either a 32-bit or 64-bit mode on a platform with addressable memory above 4 GB.

Software Implementation Note: A 64-bit OS need not require 64-bit client interface services in order to boot. Because of the problems that might be introduced by dynamically switching between 32-bit and 64-bit modes in OF, the

configuration variable **64-bit-mode?** is provided so that OF can statically configure itself to the needs of the OS.

5.1.3 Processor Interface Variations

Individual processor interface implementations are described in their respective user's manuals.

5.1.4 PA Features Deserving Comment

Some PA features are optional, and need not be implemented in a platform. Usage of others may be discouraged due to their potential for poor performance. The following sections elaborate on the disposition of these features in regard to compliance with the PA.

5.1.4.1 Multiple Scalar Operations

The PA supports multiple scalar operations. The multiple scalar operations are *Load and Store String* and *Load and Store Multiple*. Due to the long-term performance disadvantage associated with multiple scalar operations, their use by software is not recommended.

5.1.4.2 External Control Instructions (Optional)

The external control instructions (*eciwx* and *ecowx*) are not supported by this architecture.

5.1.5 cpu Node "Status" Property

See Appendix C, "PA Processor Binding," on page 753 for the values of the **"status"** property of the **cpu** node.

5.1.6 Multi-Threading Processor Option

Power processors may optionally support multi-threading.

R1–5.1.6–1. For the Multi-threading Processor option: The platform must supply one entry in the **ibm,ppc-interrupt-server#s** property associated with the processor for each thread that the processor supports.

Refer to Section B.6.4, "Properties of the Node of type cpu," on page 698 for the definition of the **ibm,ppc-interrupt-server#s** property.

5.2 Memory Architecture

The Memory Architecture of an LoPAPR implementation is defined by *Power ISA* [1] and Chapter 3, "Address Map," on page 59, which defines what platform elements are accessed by each real (physical) system address, as well as the sections which follow.

The PA allows implementations to incorporate such performance enhancing features as write-back caching, non-coherent instruction caches, pipelining, and out-of-order and speculative execution. These features introduce the concepts of *coherency* (the apparent order of storage operations to a single memory location as observed by other processors and DMA) and *consistency* (the order of storage accesses among multiple locations). In most cases, these features are transparent to software. However, in certain circumstances, OS software explicitly manages the order and buffering of storage operations. By selectively eliminating ordering options, either via storage access mode bits or the introduction

of storage barrier instructions, software can force increasingly restrictive ordering semantics upon its storage operations. Refer to *Power ISA* [1] for further details.

PA processor designs usually allow, under certain conditions, for caching, buffering, combining, and reordering in the platform's memory and I/O subsystems. The platform's memory subsystem, system interconnect, and processors, which cooperate through a platform implementation specific protocol to meet the PA specified memory coherence, consistency, and caching rules, are said to be within the platform's *coherency domain*.

Figure 10, "Example System Diagram Showing the PA Coherency Domain," on page 96 shows an example system. The shaded portion is the PA coherency domain. Buses 1 through 3 lie outside this domain. The figure shows two I/O subsystems, each interfacing with the host system via a Host Bridge. Notice that the domain includes portions of the Host Bridges. This symbolizes the role of the bridge to apply PA semantics to reference streams as they enter or leave the coherency domain, while implementing the ordering rules of the I/O bus architecture.

Memory, other than System Memory, is not required to be coherent. Such memory may include memory in IOAs.

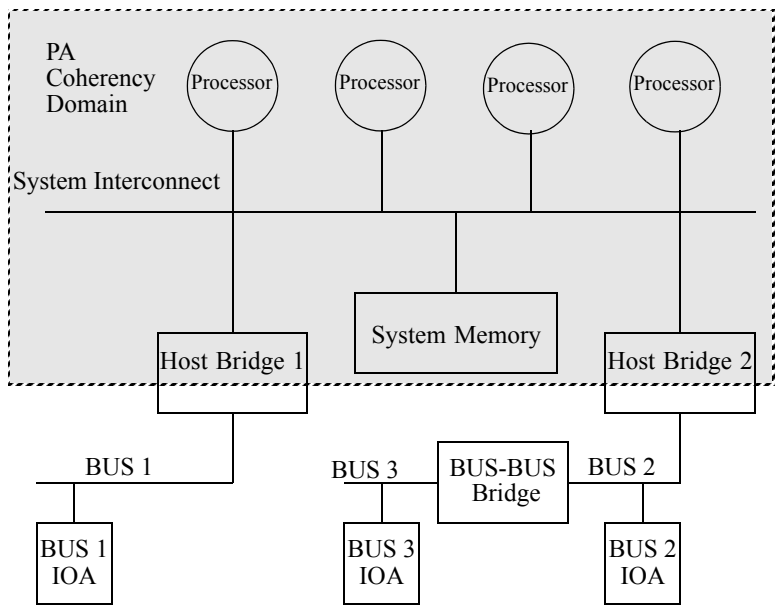


Figure 10. Example System Diagram Showing the PA Coherency Domain

Hardware Implementation Note: Components of the platform within the coherency domain (memory controllers and in-line caches, for example) collectively implement the PA memory model, including the ordering of operations. Special care should be given to configurations for which multiple paths exist between a component that accesses memory and the memory itself, if accesses for which ordering is required are permitted to use different paths.

5.2.1 System Memory

System Memory normally consists of dynamic read/write random access memory which is used for the temporary storage of programs and data being operated on by the processor(s). A platform usually provides for the expansion of System Memory via plug-in memory modules and/or memory boards.

R1–5.2.1–1. Platforms must provide at least 128 MB of System Memory. (Also see Chapter 3, "Address Map," on page 59 for other requirements which apply to memory within the first 32MB of System Memory.)

R1–5.2.1–2. Platforms must support the expansion of System Memory to 2 GB or more.

Hardware Implementation Note: These requirements are minimum requirements. Each OS has its own recommended configuration which may be greater.

Software Implementation Note: System Memory will be described by the properties of the **`memory`** node(s) of the OF device tree.

5.2.2 Memory Mapped I/O (MMIO) and DMA Operations

Storage operations which cross the coherency domain boundary are referred to as Memory Mapped I/O (MMIO) operations if they are initiated within the coherency domain, and DMA operations if they are initiated outside the coherency domain and target storage within it. Accesses with targets outside the coherency domain are assumed to be made to IOAs. These accesses are considered performed (or complete) when they complete at the IOA's I/O bus interface.

Bus bridges translate between bus operations on the initiator and target buses. In some cases, there may not be a one-to-one correspondence between initiator and target bus transactions. In these cases, the bridge selects one or a sequence of transactions which most closely matches the meaning of the transaction on the source bus. See also Chapter 4, "I/O Bridges and Topologies," on page 71 for more details and the appropriate PCI specifications.

For MMIO *Load* and *Store* instructions, the software needs to set up the WIMG bits appropriately to control *Load* and *Store* caching, *Store* combining, and speculative *Load* execution to I/O addresses. This architecture does not require platform support of caching of MMIO *Load* and *Store* instructions. See the PA for more information.

R1–5.2.2–1. For MMIO *Load* and *Store* instructions, the hardware outside of the processor must not introduce any reordering of the MMIO instructions for a processor or processor thread which would not be allowed by the PA for the instruction stream executed by the processor or processor thread.

Hardware Implementation Note: Requirement R1–5.2.2–1 may imply that hardware outside of the processor cannot reorder MMIO instructions from the same processor or processor thread, but this depends on the processor implementation. For example, some processor implementations will not allow multiple *Loads* to be issued when those *Loads* are to Cache Inhibited and Guarded space (as are MMIO *Loads*) or allow multiple *Stores* to be issued when those *Stores* are to Cache Inhibited and Guarded space (as are MMIO *Stores*). In this example, hardware external to the processors could re-order *Load* instructions with respect to other *Load* instructions or re-order *Store* instructions with respect to other *Store* instructions since they would not be from the same processor or thread. However, hardware outside of the processor must still take care not to re-order *Loads* with respect to *Stores* or vice versa, unless the hardware has access to the entire instruction stream to see explicit ordering instructions, like eieio. Hardware outside of the processor includes, but is not limited to, buses, interconnects, bridges, and switches, and includes hardware inside and outside of the coherency domain.

R1–5.2.2–2. *(Requirement Number Reserved For Compatibility)*

Apart from the ordering disciplines stated in Requirements R1–5.2.2–1 and, for PCI the ordering of MMIO *Load* data return versus buffered DMA data, as defined by Requirement R1–4.2.2.2–6, no other ordering discipline is guaranteed by the system hardware for *Load* and *Store* instructions performed by a processor to locations outside the PA coherency domain. Any other ordering discipline, if necessary, must be enforced by software via programming means.

The elements of a system outside its coherency domain are not expected to issue explicit PA ordering operations. System hardware must therefore take appropriate action to impose ordering disciplines on storage accesses entering the coherency domain. In general, a strong-ordering rule is enforced on an IOA's accesses to the same location, and write operations from the same source are completed in a sequentially consistent manner. The exception to this rule is for the special protocol ordering modifiers that may exist in certain I/O bus protocols. An example of such a protocol ordering modifier is the PCI Relaxed Ordering bit[1], as indicated in the requirements, below.

1.The PCI Relaxed Ordering bit is an optional implementation, from both the IOA and platform perspective.

R1–5.2.2–3. Platforms must guarantee that accesses entering the PA coherency domain that are from the same IOA and to the same location are completed in a sequentially consistent manner, except transactions from PCI-X and PCI Express masters may be reordered when the Relaxed Ordering bit in the transaction is set, as specified in the *PCI-X Protocol Addendum to the PCI Local Bus Specification* [21] and *PCI Express Base Specification* [22].

R1–5.2.2–4. Platforms must guarantee that multiple write operations entering the PA coherency domain that are issued by the same IOA are completed in a sequentially consistent manner, except transactions from PCI-X and PCI Express masters may be reordered when the Relaxed Ordering bit in the transaction is set, as specified in the *PCI-X Protocol Addendum to the PCI Local Bus Specification* [21] and *PCI Express Base Specification* [22].

R1–5.2.2–5. Platforms must be designed to present I/O DMA writes to the coherency domain in the order required by *Power ISA* [1], except transactions from PCI-X and PCI Express masters may be reordered when the Relaxed Ordering bit in the transaction is set, as specified in the *PCI-X Protocol Addendum to the PCI Local Bus Specification* [21] and *PCI Express Base Specification* [22].

5.2.3 Storage Ordering and I/O Interrupts

The conclusion of I/O operations is often communicated to processors via interrupts. For example, at the end of a DMA operation that deposits data in the System Memory, the IOA performing the operation might send an interrupt to the processor. Arrival of the interrupt, however, may be no guarantee that all the data has actually been deposited; some might be on its way. The receiving program must not attempt to read the data from the memory before ensuring that all the data has indeed been deposited. There may be system and I/O subsystem specific method for guaranteeing this. See Section 4.2.2.2, “PCI DMA Ordering,” on page 78.

5.2.4 Atomic Update Model

An update of a memory location by a processor, involving a *Load* followed by a *Store*, can be considered “atomic” if there are no intervening *Store*s to that location from another processor or mechanism. The PA provides primitives in the form of *Load And Reserve* and *Store Conditional* instructions which can be used to determine if the update was indeed atomic. These primitives can be used to emulate operations such as “atomic read-modify-write” and “atomic fetch-and-add.” Operation of the atomic update primitives is based on the concept of “Reservation,”[1] which is supported in an LoPAPR system via the coherence mechanism.

R1–5.2.4–1. The *Load And Reserve* and *Store Conditional* instructions must not be assumed to be supported for Write-Through storage.

Software Implementation Note: To emulate an atomic read-modify-write operation, the instruction pair must access the same storage location, and the location must have the Memory Coherence Required attribute.

Hardware Implementation Note: The reservation protocol is defined in Book II of the *Power ISA* [1] for atomic updates to locations in the same coherency domain.

R1–5.2.4–2. The *Load And Reserve* and *Store Conditional* instructions must not be assumed to be supported for Caching-Inhibited storage.

1.See Book I and II of *Power ISA* [1].

5.2.5 Memory Controllers

A *Memory Controller* responds to the real (physical) addresses produced by a processor or a host bridge for accesses to System Memory. It is responsible for handling the translation from these addresses to the physical memory modules within its configured domain of control.

R1–5.2.5–1. Memory controller(s) must support the accessing of System Memory as defined in Chapter 3, "Address Map," on page 59.

R1–5.2.5–2. Memory controller(s) must be fully initialized and set to full power mode prior to the transfer of control to the OS.

R1–5.2.5–3. All allocations of System Memory space among memory controllers must have been done prior to the transfer of control to the OS.

Software Implementation Note: Memory controller(s) are described by properties of the **`memory-controller`** node(s) of the OF device tree.

5.2.6 Cache Memory

All of the PA processors include some amount of on-chip or *internal cache* memory. This architecture allows for cache memory which is external to the processor chip, and this *external cache* memory forms an extension to internal cache memory.

R1–5.2.6–1. If a platform implementation elects not to cache portions of the address map in all external levels of the cache hierarchy, the result of not doing so must be transparent to the operation of the software, other than as a difference in performance.

R1–5.2.6–2. All caches must be fully initialized and enabled, and they must have accurate state bits prior to the transfer of control to the OS.

R1–5.2.6–3. If an in-line external cache is used, it must support one reservation as defined for the *Load And Reserve* and *Store Conditional* instructions.

R1–5.2.6–4. For the Symmetric Multiprocessor option: Platforms must implement their cache hierarchy such that all caches at a given level in the cache hierarchy can be flushed and disabled before any caches at the next level which may cache the same data are flushed and disabled (that is, L1 first, then L2, and so on).

R1–5.2.6–5. For the Symmetric Multiprocessor option: If a cache implements snarfing, then the cache must be capable of disabling the snarfing during flushing in order to implement the RTAS *stop-self* function in an atomic way.

R1–5.2.6–6. Software must not depend on being able to change a cache from copy-back to write-through.

Software Implementation Notes:

1. Each first level cache will be defined via properties of the **`cpu`** node(s) of the OF device tree. Each higher level cache will be defined via properties of the **`l2-cache`** node(s) of the OF device tree. See Appendix C, "PA Processor Binding," on page 753 for more details.

2. To ensure proper operation, cache(s) at the same level in the cache hierarchy should be flushed and disabled before cache(s) at the next level (that is, L1 first, then L2, and so on).

5.2.7 Memory Status information

New OF properties are defined to support the identification and contain the status information on good and bad system memory.

R1–5.2.7–1. Firmware must implement all of the properties for memory modules, as specified by Appendix B, "LoPAPR Binding," on page 661, and any other properties defined by this document which apply to memory modules.

5.2.8 Reserved Memory

Sections of System Memory may be reserved for usage by OS extensions, with the restrictions detailed below. Memory nodes marked with the special value of the **"status"** property of "reserved" is not to be used or altered by the base OS. Several different ranges of memory may be marked as "reserved". If DLPAR of memory is to be supported and growth is expected, then, an address range must be unused between these areas in order to allow growth of these areas. Each area has its own DRC Type (starting at 0, MEM, MEM-1, MEM-2, and so on). Each area has a current and a maximum size, with the current size being the sum of the sizes of the populated DRCs for the area and the max being the sum total of the sizes of all the DRCs for that area. The logical address space allocated is the size of the sum of the all the areas' maximum sizes. Starting with logical real address 0, the address areas are allocated in the following order: OS, DLPAR growth space for OS (if DLPAR is supported), reserved area (if any) followed by the DLPAR growth space for that reserved area (if DLPAR is supported), followed by the next reserved space (if any), and so on. The current memory allocation for each area is allocated contiguously from the beginning of the area. On a boot or reboot, including hypervisor reboot, if there is any data to be preserved (that is, the **"ibm,preserved-storage"** property exists in the RTAS node), then the starting logical real address of each LMB is maintained through the reboot. The memory in each region can be independently increased or decreased using DLPAR memory functions, when DLPAR is supported. Changes to the current memory allocation for an area results in the addition or removal of memory to the end of the existing memory allocation.

Implementation Note: if the shared memory regions are not accessed by the programs, and are just used for DMA most of the time, then the same HPFT hit rate could be achieved with a far lower ration of HPFT entries to logical storage space.

R1–5.2.8–1. For the Reserved Memory option: Memory nodes marked with the special value of the **"status"** property of "reserved" must not be used or altered by the base OS

Implementation Note: How areas get chosen to be marked as reserved is beyond the scope of this architecture.

R1–5.2.8–2. For the Reserved Memory option with the LRDR option: Each unique memory area that is to be changed independently via DLPAR must have different DRC Types (for example, MEM, MEM-1, and so on).

5.2.9 Persistent Memory

Selected regions of storage (LMBs) may be optionally preserved across client program boot cycles. See Section 2.1.3.6.12, "Persistent Memory and Memory Preservation Boot (Storage Preservation Option)," on page 49 and Section 7.4.4, "Managing Storage Preservation," on page 240.

6 Interrupt Controller

This chapter specifies the requirements for the LoPAPR interrupt controller. Platforms may chose to virtualize the interrupt controller or to provide the PowerPC External Interrupt option.

6.1 Interrupt Controller Virtualization

Virtualization of the interrupt controller is done through the Interrupt Support hcalls. See Section 14.5.4.7, “Interrupt Support hcall()s,” on page 429.

6.2 PowerPC External Interrupt Option

The PowerPC External Interrupt option is based upon a subset of the PowerPC External Interrupt Architecture. The PowerPC External Interrupt Architecture contains a register-level architectural definition of an interrupt control structure. This architecture defines means for assigning properties such as priority, destination, etc., to I/O and interprocessor interrupts, as well as an interface for presenting them to processors. It supports both specific and distributed methods for interrupt delivery. See also Appendix A, “PowerPC External Interrupt Architecture,” on page 1015.

In NUMA platform configurations, the interrupt controllers may be configured in disjoint domains. The firmware makes the server numbers visible to any single OS image appear to come from a single space without duplication. This may be done by appropriately initializing the interrupt presentation controllers or the firmware may translate the server numbers presented to it in RTAS calls before entering them into the interrupt controller registers. The OS is made aware that certain interrupts are only served by certain servers by the inclusion of the **`“ibm,interrupt-domain”`** property in the interrupt controller nodes.

6.2.1 PowerPC External Interrupt Option Requirements

The following are the requirements for the PowerPC External Interrupt option. Additional requirements and information relative to the MSI option, when implemented with this option, are listed in Section 6.2.3, “MSI Option,” on page 103.

R1–6.2.1–1. For the PowerPC External Interrupt option: Platforms must implement interrupt architectures that are in register-level architectural compliance with Appendix A, “PowerPC External Interrupt Architecture,” on page 1015.

R1–6.2.1–2. For the PowerPC External Interrupt option: The platform’s OF device tree must include one or more PowerPC External Interrupt Presentation node(s), as children of the root node.

R1–6.2.1–3. For the PowerPC External Interrupt option: The platform’s OF device tree must include an **`“ibm,ppc-interrupt-server#s”`** and an **`“ibm,ppc-interrupt-gserver#s”`** property as defined for each processor in the processor’s **`/cpus/cpu`** node.

R1–6.2.1–4. For the PowerPC External Interrupt option: The various **`“ibm,ppc-interrupt-server#s”`** property values seen by a single OS image must be all unique.

R1–6.2.1–5. For the PowerPC External Interrupt option: If an OS image sees multiple global interrupt server queues, the **"ibm,ppc-interrupt-gserver#s"** properties associated with the various queues must have unique values.

R1–6.2.1–6. For the PowerPC External Interrupt option: The platform's OF device tree must include a PowerPC External Interrupt Source Controller node, as defined for each Bus Unit Controller (BUC) that can generate PowerPC External Interrupt Architecture interrupts, as a child of the platform's root node.

R1–6.2.1–7. For the PowerPC External Interrupt option: The platform's OF device tree must conform to the *Open Firmware: Recommended Practice - Interrupt Mapping [7]* and include the appropriate mapping and interrupt properties to allow the mapping of all non-zero XISR values (*interrupt#*) to the corresponding node generating the interrupt.

R1–6.2.1–8. For the PowerPC External Interrupt option: The PowerPC External Interrupt Presentation Controller node must not contain the **"used-by-rtas"** property.

R1–6.2.1–9. For the PowerPC External Interrupt option: The PowerPC External Interrupt Source Controller node must contain the **"used-by-rtas"** property.

R1–6.2.1–10. For the PowerPC External Interrupt option: If the interrupt hardware is configured such that, viewed from any given OS image, any interrupt source controller cannot direct interrupts to any interrupt presentation controller, then the platform must include the **"ibm,interrupt-domain"** property in all interrupt source and presentation controller nodes for that OS so that the OS can determine the servers that may be valid targets for any given interrupt.

R1–6.2.1–11. For the PowerPC External Interrupt option: All interrupt controller registers must be accessed via Caching-Inhibited, Memory Coherence not required and Guarded Storage mapping.

R1–6.2.1–12. For the PowerPC External Interrupt option: The platform must manage the Available Processor Mask Register so that global interrupts (server number field of the eXternal Interrupt Vector Entry (XIVE) set to a value from **"ibm,ppc-interrupt-gserver#s"**) are only sent to one of the active processors.

R1–6.2.1–13. For the PowerPC External Interrupt option: The platform must initialize the interrupt priority in each XIVE to the least favored level (0xFF), enable any associated IER bit for interrupt sources owned by the OS, and set the Current Processor Priority Register to the Most favored level (0x00) prior to the transfer of control to the OS so that no interrupts are signaled to a processor until the OS has taken explicit action.

R1–6.2.1–14. For the PowerPC External Interrupt option: Any implemented PowerPC External Interrupt Architecture registers that are not reported in specific interrupt source or destination controller nodes (such as the APM register) must be included in the **"reg"** property of the **/reserved** node.

R1–6.2.1–15. For the PowerPC External Interrupt option: The interrupt source controller must prevent signalling new interrupts when the XIVE interrupt priority field is set to the least favored level.

R1–6.2.1–16. For the PowerPC External Interrupt option: Interrupt controllers that do not implement the behavior of Requirement R1–6.2.1–15, must provide an Interrupt Enable Register (IER) which can be manipulated by RTAS,

R1–6.2.1–17. For the PowerPC External Interrupt option: The platform must assign the Bus Unit Identifiers (BUIDs) such that they form a compact address space. That is, while the first BUID value is arbitrary, subsequent BUIDs should be contiguous.

R1–6.2.1–18. For the PowerPC External Interrupt option: Platforms implementing interrupt server number fields greater than 8 bits must include the **"ibm,interrupt-server#-size"** property in the interrupt source controller node.

R1–6.2.1–19. For the PowerPC External Interrupt option: Platforms implementing interrupt buid number fields greater than 9 bits must include the **`"ibm,interrupt-buid-size"`** property in the interrupt presentation controller node.

R1–6.2.1–20. For the PowerPC External Interrupt option: Platforms must include the **`"ibm,interrupt-server-ranges"`** property in the interrupt presentation controller node.

6.2.2 PowerPC External Interrupt Option Properties

See Appendix B, “LoPAPR Binding,” on page 661 for property definitions.

6.2.3 MSI Option

The Message Signaled Interrupt (MSI) or Enhanced MSI (MSI-X) capability of PCI IOAs in many cases allows for greater flexibility in assignment of external interrupts to IOA functions than the predecessor Level Sensitive Interrupt (LSI) capability, and in some cases treats MSIs as a resource pool that can be reassigned based on availability of MSIs and the need of an IOA function for more interrupts than initially assigned. Platforms that implement the MSI option implement the *ibm,change-msi* and *ibm,query-interrupt-source-number* RTAS calls. These RTAS calls manage interrupts in a platform that implements the MSI option. In particular, these calls assign additional MSI resources to an IOA function (as defined by its PCI configuration address: *PHB_Unit_ID_Hi, PHB_Unit_ID_Low, and config_addr*), when supported by the platform. See Section 7.3.10.5, “MSI Support,” on page 172 for more information on theses RTAS calls for MSI management.

This architecture will refer generically to the MSI and MSI-X capabilities as simply “MSI,” except where differentiation is required. In this architecture, MSIs and LSIs are what the IOA function signals, and what the software sees for that signal is ultimately the LSI or MSI *source number.* The interrupt source numbers returned by the *ibm,query-interrupt-source-number* RTAS call are the numbers used to control the interrupt as in the *ibm,get-xive*, *ibm,set-xive*, *ibm,int-on*, and *ibm,int-off* RTAS calls.

PCI-X and PCI Express IOA functions that signal interrupts are required by the PCI specifications to implement either the MSI or MSI-X interrupt capabilities, or both. For PCI Express, it is expected that IOAs will only support MSI or MSI-X (that is, no support for LSIs). When both MSI and MSI-X are implemented by an IOA function, the MSI method will be configured by the platform, but may be overridden by the OS or device driver, via the *ibm,change-msi* RTAS call, to be MSI-X or, if assigned by the firmware, to LSI (by removal of the MSIs assigned). Table 14, “LSI and MSI Support Requirements and Initial Assignment,” on page 103 summarizes the LSI and MSI support.

Table 14. LSI and MSI Support Requirements and Initial Assignment

IOA Type	LSI required by PCI specifications?	MSI or MSI-X required by PCI specifications?	Bridge between IOA and PHB	Possible platform support		Initial interrupt assignment[a]
				If PHB does not support MSI option (Not including PCI Express HBs)	If PHB supports MSI option (Including all PCI Express HBs)	
PCI	When interrupts are required	No (allowed; optional)	-	LSI	LSI or MSI	LSI[b]
PCI-X	Encouraged when interrupts are required, for backward platform compatibility	Yes	-	LSI	LSI or MSI	LSI[c]

Table 14. LSI and MSI Support Requirements and Initial Assignment ***(Continued)***

IOA Type	LSI required by PCI specifications?	MSI or MSI-X required by PCI specifications?	Bridge between IOA and PHB	Possible platform support		Initial interrupt assignment[a]
				If PHB does not support MSI option (Not including PCI Express HBs)	If PHB supports MSI option (Including all PCI Express HBs)	
PCI Express	Discouraged (expect IOAs to not implement in most cases)	Yes	None or PCI Express switch only	n/a	MSI	MSI[d]
			Reverse bridge (primary, PCI Express secondary)	LSI or not supported[e]	LSI[f] or MSI	LSI[g]

a. Assignment means to allocate the platform resources and to enable the interrupt in the IOA function's configuration space.
b. If MSIs are to be supported, the device driver must enable via the *ibm,change-msi* RTAS call.
c. If MSIs are to be supported, the device driver must enable via the *ibm,change-msi* RTAS call.
d. MSI as an initial assignment means that one or more MSIs are reported as being available for the IOA function. In addition, LSIs may also be reported but not enabled, in which case if the device driver removes the assigned MSIs, the assigned LSI are enabled by the platform firmware in the IOA function's configuration space.
e. If PCI Express IOA function does not support LSI, then this combination is not supported.
f. If PCI Express IOA function does not support LSI, then this combination is not supported.
g. If the PCI Express IOA function does not support LSI, then the platform will set the initial interrupt assignment to MSI, and if the device driver does not support MSI, then the IOA function will not be configurable (that is, conversion from MSI to LSI through the bridge is not supported by this architecture). If LSI is the initial assignment, then if MSIs are to be supported, device driver must enable via the *ibm,change-msi* RTAS call.

The *ibm,change-msi* RTAS call is used to query the initial number of MSIs assigned to a PCI configuration address and to request a change in the number of MSIs assigned. The MSIs interrupt source numbers assigned to an IOA function are returned via the *ibm,query-interrupt-source-number* RTAS call. In addition, when the *ibm,query-interrupt-source-number* RTAS call is implemented, it may be used to query the LSI source numbers, also. The *ibm,query-interrupt-source-number* RTAS call is called iteratively, once for each interrupt assigned to the IOA function. When an IOA function receives an initial assignment of an LSI, the interrupt number for that LSI may also be obtained through the same OF device tree properties that are used to report interrupt information when the *ibm,query-interrupt-source-number* RTAS call is not implemented.

R1–6.2.3–1. The platform must implement the MSI option if the platform contains at least one PCI Express HB.

Architecture and Software Note: The MSI option may also be implemented in the absence of any PCI Express HBs. In that case, the implementation of the MSI option is via the presence of the implementation of the associated *ibm,change-msi* and *ibm,query-interrupt-source-number* RTAS calls.

R1–6.2.3–2. For the MSI option: The platform must implement the PowerPC External Interrupt option.

R1–6.2.3–3. For the MSI option: The platform must implement the *ibm,change-msi* and *ibm,query-interrupt-source-number* RTAS calls.

R1–6.2.3–4. For the MSI option: The platform must initially assign LSI or MSIs to IOA functions as defined in Table 14, "LSI and MSI Support Requirements and Initial Assignment," on page 103 and must enable the assigned interrupts in the IOA function's configuration space (the interrupts remains disabled at the PHB, and must be enabled by the device driver though the *ibm,set-xive* and *ibm,int-on* RTAS calls.

R1–6.2.3–5. For the MSI option: The platform must provide a minimum of one MSI per IOA function (that is per each unique PCI configuration address, including the Function #) to be supported beneath the interrupt source controller, and any given MSI and MSI source number must not be shared between functions or within one function (even within the same PE).

R1–6.2.3–6. For the MSI option: The platform must provide at least one MSI port (the address written by the MSI) per Partitionable Endpoint (PE).

Platform Implementation Note: Requirement R1–6.2.3–5 in conjunction with Requirement R1–6.2.3–6 may have certain ramifications on the design. Depending on the implementation, a unique MSI port per IOA function may be required, and not just a unique port per PE.

R1–6.2.3–7. For the MSI option with the LPAR option: The platform must prevent a PE from creating an interrupt to a partition other than those to which the PE is authorized by the platform to interrupt.

R1–6.2.3–8. For the MSI option: The platform must set the PCI configuration space MSI registers properly in an IOA at all the following times:

a. Initial boot time

b. During the *ibm,configure-connector* RTAS call

c. During the *ibm,change-msi* or *ibm,query-interrupt-source-number* RTAS call

R1–6.2.3–9. For the MSI option: The platform must initialize any bridges necessary to appropriately route interrupts at all the following times:

a. At initial boot time

b. During the *ibm,configure-connector* RTAS call

c. During the *ibm,configure-bridge* RTAS call

d. During the *ibm,change-msi* or *ibm,query-interrupt-source-number* RTAS call

R1–6.2.3–10. For the MSI option: The platform must provide the **`"ibm,req#msi"`** property for any IOA function which is requesting MSIs; at initial boot time and during the *ibm,configure-connector* RTAS call.

R1–6.2.3–11. For the MSI option: The platform must remember and recover on error recovery any previously allocated and setup interrupt information in the platform-owned hardware.

Software and Platform Implementation Note: In Requirement R1–6.2.3–11, it is possible that some interrupts may be lost as part of the error recovery, and software should be implemented to take into consideration that possibility.

6.3 Platform Reserved Interrupt Priority Level Option

The Platform Reserved Interrupt Priority Level option allows platforms to reserve interrupt priority levels for internal uses. When the platform exercises this option, it notifies the client program via the OF device tree **`"ibm,plat-res-int-priorities"`** property of the **`root`** node of the device tree.

R1–6.3–1. For the Platform Reserved Interrupt Priority Level option: The platform must include the **`"ibm,plat-res-int-priorities"`** property in the **`root`** node of the device tree.

R1–6.3–2. For the Platform Reserved Interrupt Priority Level option: The platform must not reserve priority levels 0x00 through 0x07 and 0xFF for internal use.

7 Run-Time Abstraction Services

7.1 RTAS Introduction

The Run-Time Abstraction Service (RTAS) functions are provided by LoPAPR platforms to insulate the OS from having to know about and manipulate a number of key platform hardware functions which ordinarily require platform-dependent code. The OS calls RTAS functions rather than manipulating hardware registers directly, reducing the need for platform tailoring by the OS. This method of abstracting access to these platform functions also permits hardware designers considerable flexibility in hardware implementation. Since RTAS is provided by the platform developer, this approach places the responsibility for supporting the platform hardware design with the platform developer, not the OS developer. This permits a degree of independence between the schedules of hardware and software and reduces the release and test requirements for the OS, since it can be tested to conform to the RTAS interface and not to every specific hardware implementation. See Table 17, "RTAS Tokens for Functions," on page 112 for a list of all RTAS calls, and which ones are required based on which LoPAPR options that are implemented in the platform.

In order for platforms to achieve this separation of OS code from hardware implementation dependencies, RTAS defines an interface between the platform and the OS that provides control of some of the common devices found on all platforms. RTAS is a system programming interface that is realized, on a specific platform, by an RTAS implementation. The RTAS implementation provides the platform specific processing of the common components. RTAS limits itself to the run-time control of non-I/O, typically system board-resident, hardware features. Traditionally, features such as these have been implemented differently on different platforms. The different implementations have required much effort and platform-dependent code in the OS. RTAS permits the OS to operate over a much wider range of platforms without specialized code for each platform.

In general, the OS should not access RTAS resources directly. It should call RTAS to control the resource.

OS drivers are necessary to provide device specific processing for IOAs.

The role of RTAS versus OF is very important to understand. OF and RTAS are both platform-specific software, and both are tailored by the platform developer to manipulate the specific platform hardware. However, RTAS is intended to be present *during* the execution of the OS, and to be called by the OS to access platform hardware features on behalf of the OS, whereas OF *need not* be present when the OS is running. This frees OF's memory to be used by applications. RTAS is small enough to painlessly coexist with the OS and applications.

This chapter uses the term RTAS to refer both to the architected RTAS interface and to an RTAS implementation.

7.2 RTAS Environment

RTAS provides an interface definition between the OS and the firmware provided by the platform. This chapter defines the calling conventions used by both the OS and the platform's RTAS firmware.

RTAS runs with instruction and data relocate as well as processing exceptions disabled. To not interfere with the OS, RTAS may not cause any exceptions, nor can it depend on any particular virtual memory mappings.

All RTAS functions are invoked from the OS by calling the *rtas-call* function. The address of this function is obtained from OF when RTAS is instantiated. See Requirement R1–7.2.3–1 for more details. RTAS determines what function to

invoke based on the data passed into the *rtas-call* function. This section describes the mechanisms used to invoke the *rtas-call* function, the machine state, register usage, resource allocation, and the invocation requirements.

If the LPAR option is enabled, multiple partitions may exist, each with its own OS instance. This requires some changes to the RTAS environment. These changes are discussed in Chapter 14, "Logical Partitioning Option," on page 385

7.2.1 Machine State

When RTAS functions are invoked, the calling processor shall have address translations, floating point, and most other exceptions disabled and it shall be running in privileged state.

R1–7.2.1–1. RTAS must be called in "real mode," that is, all address translation must be disabled. Bits MSR_{IR} and MSR_{DR} of the MSR register must be zero.

R1–7.2.1–2. RTAS must be called in privileged mode, and the MSR_{PR} bit must be set to 0.

R1–7.2.1–3. RTAS must be called with external interrupts disabled, and the MSR_{EE} bit must be set to 0.

R1–7.2.1–4. RTAS must be called with trace disabled, and the MSR_{SE} and MSR_{BE} bits must be set to 0.

R1–7.2.1–5. RTAS must be called with floating point disabled, and the MSR_{FE0}, MSR_{FE1}, and MSR_{FP} bits must be set to 0.

R1–7.2.1–6. RTAS must be called with the MSR_{SF}, (MSR_{ISF}, and ASR_{V} bits if applicable on the specific processor) set to match the mode used to instantiate RTAS (0 for **`instantiate-rtas`** or 1 for **`instantiate-rtas-64`**) and the LE bit set to 0.

R1–7.2.1–7. With the exception of the MSR_{DR} and MSR_{RI} bits, RTAS must not change the state of the machine by modifying the MSR.

R1–7.2.1–8. If *rtas-call* is entered in a non-recoverable mode, indicated by having the MSR_{RI} bit set equal to 0, then RTAS must not enter a recoverable mode by setting the MSR_{RI} bit to 1.

R1–7.2.1–9. If called with MSR_{RI} equal to 1, then RTAS must protect its own critical regions from recursion by setting the MSR_{RI} bit to 0 when in the critical regions.

Software Implementation Notes:

1. If the MSR_{ME} bit is left enabled, the OS's exception handler must be aware that RTAS might have been running and that various processor registers might not be in the expected state for an interrupted OS, which precludes recoverability but permit logging machine checks.
2. There are some provisions for recursive calls to RTAS error handling functions. Therefore, RTAS should set the MSR_{RI} bit to 0 if SRR0/SRR1 or any other RTAS resource is in a state where information could be lost and prohibit recovery.
3. Requirement R1–7.2.1–6 implies that RTAS must be able to be instantiated in 64-bit mode on platforms that can support 64-bit execution.

7.2.2 Register Usage

R1–7.2.2–1. Except as required by a specific function, registers SPRG2, R0, R3 through R12, CTR, XER, LR, and fields CR2-CR4 of the CR, RTAS must preserve all OS visible register state.

R1–7.2.2–2. RTAS must not modify the DEC and registers SPRG0, SPRG1, and SPRG3.

Software Implementation Notes:

1. RTAS is entered in real mode (with address translation turned off). In this mode, all data accesses are assumed to be cached in copy back mode with memory coherence required. Since these settings may not be appropriate for all accesses, RTAS is free to transparently use the processor specific facilities required to access platform hardware resources. The OS machine check handler can only depend on those registers that are required to be unchanged (see Requirement R1–7.2.2–1).

2. RTAS must not change the preserved registers, or must first save them and restore the original contents before returning to the OS.

3. The SRR0-SRR1, LR, CTR, XER registers, as well as any reservations made via the load and reserve instructions, need not be preserved.

7.2.3 RTAS Critical Regions

The OS, when using RTAS, is responsible for protecting RTAS and devices used by RTAS from any simultaneous accesses that could corrupt memory or device registers. Such corruption could be caused by simultaneous execution of RTAS code, or by a device driver accessing a control register that is also modified by RTAS. In a single processor system, since the MSR_{EE} bit is 0 when entering RTAS, a call to RTAS while it is in execution is prevented except for the machine check handler. This handler may need to call various RTAS services such as *check-exception* or *system-reboot* even if the error was detected while in an RTAS service.

The OS and RTAS must co-exist on the same platform. RTAS must not change device registers that are used by the OS, nor may the OS change device registers on devices used by RTAS. With the advent of more and more integration into common super parts, some of these registers may physically reside on the same component. In this section, device implies the collection of common registers that together perform a function. Each device must be represented in the OF device tree.

R1–7.2.3–1. Except as noted in Requirement R1–7.2.3–7 and R1–7.2.3–8, the OS must ensure that RTAS is not called while RTAS is in execution and that RTAS is not simultaneously called from different processors in a multi-processor system.

R1–7.2.3–2. Any RTAS access to device or I/O registers specified in this document must be made in such a way as to be transparent to the OS.

R1–7.2.3–3. Any device that is used to implement the RTAS abstracted services must have the property **`"used-by-rtas"`** in the OF device tree. However, if the device is only used by the *power-off,* and *system-reboot* calls, the property should not be set. The *rtas-display-device* must be marked with the property **`"used-by-rtas"`** if it is a specialized device only to be accessed via the RTAS *display-character* call and not otherwise shared with the OS

Software Implementation Note: Table 15, “Use of “used-by-rtas”,” on page 109 clarifies when a device should be marked with the **`"used-by-rtas"`** property, based on whether it has any interaction with RTAS and/or the OS (with the exception of the calls listed in Requirement R1–7.2.3–1).

Table 15. Use of **`"used-by-rtas"`**

	Normal Device (1)	rtas-display-device
Only used by OS	not marked	N/A (2)
Only used by RTAS	marked	marked
OS and RTAS shared	marked (3)	not marked (4)

Table 15. Use of **"used-by-rtas"**

	Normal Device (1)	rtas-display-device
Virtual	N/A	N/A

1. A device which is not normally to be used by the OS must meet one of the following rules.
   a. It must not be included in the OF device tree.
   b. It must be defined as a "reserved" device node.
   c. It must be marked with the **"used-by-rtas"** property in the OF device tree.
2. It is assumed that an *rtas-display-device* is used by RTAS.
3. It is assumed that there are no devices other than the *rtas-display-device* which are used by both RTAS and an "unaware" OS. To allow an aware OS to share a device with RTAS, the device should be marked.
4. It is assumed that the *rtas-display-device* is used by both RTAS and the OS (as coordinated by the OS via display-character) unless it is marked. See also Requirement R1–7.3.5.3–14.

R1–7.2.3–4. Platforms must be designed such that accesses to devices that are marked **"used-by-rtas"** have no side effects on other registers in the system.

R1–7.2.3–5. Any OS access to devices specified as **"used-by-rtas"** must be made in such a way as to be transparent to RTAS.

R1–7.2.3–6. RTAS must not generate any exceptions (for example, no alignment exceptions, page table walk exceptions, etc.).

R1–7.2.3–7. The OS machine check and soft reset handlers must be able to call the RTAS services:

- *nvram-fetch*
- *nvram-store*
- *check-exception*
- *display-character*
- *system-reboot*
- *set-power-level(0,0)*
- *power-off*
- *ibm,set-eeh-option*
- *ibm,set-slot-reset*
- *ibm,read-slot-reset-state2*

R1–7.2.3–8. The *stop-self* service need only be serialized with calls to the *stop-self*, *start-cpu*, and *set-power-level* services. The OS must be able to call RTAS services on other processors while a processor is stopped or being stopped.

Software Implementation Notes:

1. While RTAS must not generate any exceptions, it is still possible that a machine check interrupt may occur during the execution of an RTAS function. In this case, the machine check handler may be entered prior to the normal termination of the RTAS function.

2. It is permissible for the OS exception handler to make an RTAS call as long as Requirements R1–7.2.2–1 and R1–7.2.3–1 are met. In particular, it is expected that the RTAS *check-exception* is called from the OS exception handler.

7.2.4 Resource Allocation and Use

During execution, RTAS requires memory for both code and data. This memory may be in RAM, in a private memory area only known by the system firmware, or in memory allocated by the OS for RTAS use. RTAS should use this memory for its stack and any state savings. This memory is subsequently called the "RTAS private data area."

R1–7.2.4–1. The OS must allocate **"rtas-size"** bytes of contiguous real memory as RTAS private data area. This memory must be aligned on a 4096 byte boundary and may not cross a 256 MB boundary.

R1–7.2.4–2. The RTAS private data area must not be accessed by the OS.

R1–7.2.4–3. Except for the RTAS private data area, the argument buffer, System Memory pointed to by any reference parameter in the argument buffer, and any other System Memory areas explicitly permitted in this chapter, RTAS must not modify any System Memory. RTAS may, however, modify System Memory during error recovery provided that such modifications are transparent to the OS.

R1–7.2.4–4. RTAS calls may not sleep in any fashion nor busy wait for more than a very short period of time, except for *power-off, ibm,power-off-ups, set-power-level* (0,0), *system-reboot, ibm,update-flash-64-and-reboot, and ibm,os-term*.

Software Implementation Note: An RTAS call should take the same amount of time to perform a service that it would take the OS to perform the same function. A specific goal is that RTAS primitives should take less than a few tens of microseconds.

R1–7.2.4–5. For RTAS calls which do not allow the *Status* of -2 (Busy), there may be "rare" instances where an anomaly may occur and the call may take longer than a "very short period of time." In these cases, the call must complete within 250 microseconds. Otherwise, a hardware error response must be given.

7.2.5 Instantiating RTAS

RTAS is instantiated by an explicit client interface service call into OF. The OF device tree contains a property (**"rtas-size"**, under the **/rtas** node) which defines how much real memory RTAS requires from the OS. The OS allocates **"rtas-size"** bytes of real memory, and then invokes the **instantiate-rtas** or **instantiate-rtas-64** method of the **/rtas** node, passing the real address of the private data area (or zero, if **"rtas-size"** is zero) as the *rtas-base-address* input argument. Firmware binds RTAS to that address, binds the addresses of devices that RTAS uses, performs any RTAS initialization, and returns the address of the *rtas-call* function that is appropriate.

R1–7.2.5–1. The **instantiate-rtas** or **instantiate-rtas-64** OF method must have the arguments specified in Table 16, "instantiate-rtas or instantiate-rtas-64 Argument Call Buffer," on page 112.

Table 16. **`instantiate-rtas`** or **`instantiate-rtas-64`** Argument Call Buffer

Parameter Type	Name	Values
In	*rtas-base-address*	Real Address of RTAS area or zero, if **`"rtas-size"`** is zero
Out	*rtas-call*	Real address used to invoke RTAS functions

R1–7.2.5–2. The RTAS code bound and initialized by the **`instantiate-rtas`** method on a 64-bit capable platform, must be able to handle platform resources that are accessed using 64-bit addresses.

7.2.6 RTAS Device Tree Properties

The OF device tree contains a **`/rtas`** device node that describes the implemented RTAS features and the output device supported by RTAS. Within this device node are properties that describe the RTAS functions implemented by the firmware. For every implemented RTAS function, the **`/rtas`** node contains an OF property whose name is the same as the RTAS function. The value of this property is the token argument passed into the *rtas-call* function when making that specific RTAS call. Note that some RTAS functions are optional and some are required. This is defined in Table 17, "RTAS Tokens for Functions," on page 112.

R1–7.2.6–1. The OF device tree must contain a device node named **`/rtas`** which describes the RTAS implementation.

R1–7.2.6–2. The **`/rtas`** device node must have a property for each implemented RTAS function in Table 17, "RTAS Tokens for Functions," on page 112. The value of this property is a token that is passed into the *rtas-call* function to indicate which RTAS function to invoke.

Table 17. RTAS Tokens for Functions

RTAS property/function	Required?	Notes
`"nvram-fetch"` 7.3.1.1 on page 120	Required	Execution time proportional to amount of data
`"nvram-store"` 7.3.1.2 on page 121	Required	Execution time proportional to amount of data
`"get-time-of-day"` 7.3.2.2 on page 122	Required	
`"set-time-of-day"` 7.3.2.3 on page 123	Required	
`"set-time-for-power-on"` 7.3.2.4 on page 124		
`"event-scan"` 7.3.3.1 on page 125	Required	
`"check-exception"` 7.3.3.2 on page 127	Required	
`"rtas-last-error"` 7.3.3.3 on page 128	Required	
`"ibm,platform-dump"` 7.3.3.4.1 on page 129	If the Platform Dump option is implemented	

Table 17. RTAS Tokens for Functions *(Continued)*

RTAS property/function	Required?	Notes
"ibm,read-pci-config" 7.3.4.1 on page 134	Required	
"ibm,write-pci-config" 7.3.4.2 on page 135		
"display-character" 7.3.5.3 on page 140		
"set-indicator" 7.3.5.4 on page 142	Required	Some specific indicators are required, and some are optional
"get-sensor-state" 7.3.5.5 on page 145		
"ibm,set-system-parameter" 7.3.16.2 on page 212		
"ibm,get-system-parameter" 7.3.16.1 on page 211		
"set-power-level" 7.3.6.1 on page 151	Required for DR operations (see Chapter 13, "Dynamic Reconfiguration (DR) Architecture," on page 355)	
"get-power-level" 7.3.6.2 on page 152		
"power-off" 7.3.6.3 on page 153		Provided for platforms with software controlled power off capability
"ibm,power-off-ups" 7.3.6.4 on page 154	If there may be a platform controlled UPS.	For power off control in a platform which may have power backed up with an Uninterruptible Power Supply (UPS).
"system-reboot" 7.3.7.1 on page 155	Required	
"ibm,update-flash-64-and-reboot" 7.3.7.2 on page 156		
"ibm,manage-flash-image" 7.3.7.4 on page 158		
"ibm,validate-flash-image" 7.3.7.5 on page 159		
"ibm,activate-firmware" 7.3.7.6 on page 161		
"stop-self" 7.3.8.1 on page 162	See Note	
"start-cpu" 7.3.8.2 on page 162		
"query-cpu-stopped-state" 7.3.8.3 on page 164		
"ibm,os-term" 7.3.9.1 on page 165		
"ibm,exti2c" 7.3.9.2 on page 166		

Table 17. RTAS Tokens for Functions ***(Continued)***

RTAS property/function	Required?	Notes
"ibm,get-xive" 7.3.10.1 on page 169	Required for the PowerPC External Interrupt option	
"ibm,set-xive" 7.3.10.2 on page 170		
"ibm,int-off" 7.3.10.3 on page 170		
"ibm,int-on" 7.3.10.4 on page 171		
"ibm,configure-connector" 13.5.3.5 on page 369	Required for all DR options	See Chapter 13, "Dynamic Reconfiguration (DR) Architecture," on page 355
"ibm,set-eeh-option" 7.3.11.1 on page 180	Required for EEH option	
"ibm,set-slot-reset" 7.3.11.2 on page 182		
"ibm,read-slot-reset-state"	Being replaced by *ibm,read-slot-reset-state2*	See Section 7.3.11.3, "ibm,read-slot-reset-state2," on page 184.
"ibm,read-slot-reset-state2" 7.3.11.3 on page 184	Required for all platforms	
"ibm,get-config-addr-info2" 7.3.11.4 on page 187	Required on all platforms	
"ibm,slot-error-detail" 7.3.11.5 on page 188	Required for the EEH option	
"ibm,open-errinjct" 7.3.13 on page 197	Required for ERRINJCT option	
"ibm,errinjct" 7.3.13 on page 197		
"ibm,close-errinjct" 7.3.13 on page 197		
"ibm,nmi-register" 7.3.14 on page 204	Required for FWNMI option	
"ibm,nmi-interlock" 7.3.14 on page 204		
"ibm,configure-bridge" 7.3.12.1 on page 193	Required for the EEH option	
"ibm,configure-pe" 7.3.12.2 on page 195	Required for the EEH option	
"ibm,get-indices" 7.3.16.21–4 on page 234	Sometimes (see Chapter 16, "Service Indicators," on page 511)	
"ibm,get-vpd" 7.4.3 on page 238	Required for the Dynamic VPD option	
"ibm,manage-storage-preservation" 7.4.4 on page 240	Required for the Storage Preservation option	

Table 17. RTAS Tokens for Functions *(Continued)*

RTAS property/function	Required?	Notes
"ibm,get-dynamic-sensor-state" 7.4.2 on page 237	See Requirement R1–7.4.2–1	
"ibm,set-dynamic-indicator" 7.4.1 on page 236	See Requirement R1–7.4.1–1	
"ibm,change-msi" 7.3.10.5.1 on page 172	Required for the MSI option. Required if any PCI Express HB in the platform.	
"ibm,suspend-me" 7.4.6 on page 243	Required for the Partition Suspension option	
"ibm,update-nodes" 7.4.7 on page 246	Required for the Update OF Tree option	
"ibm,update-properties" 7.4.8 on page 249	Required for the Update OF Tree option	
"ibm,configure-kernel-dump" 7.4.9 on page 255	Required for the Configure Platform Assisted Kernel Dump option	
"ibm,query-pe-dma-window" 7.4.10.1 on page 260	Required for the Dynamic DMA Window (DDW) option	
"ibm,create-pe-dma-window" 7.4.10.2 on page 261		
"ibm,remove-pe-dma-window" 7.4.10.3 on page 262		
"ibm,reset-pe-dma-windows" 7.4.10.3 on page 262	Required for the LoPAPR version l of the Dynamic DMA Window (DDW) option	

Note: These commands are required in SMP platforms if dynamic reconfiguration is required of the processors. Similarly, a degraded mode may need these, or similar commands in the case of detection of excessive errors. In the case of a processor deconfigured by dynamic reconfiguration or due to excessive errors, the returned *CPU_status* from the *query-cpu-stopped-state* RTAS call is 2 (The processor thread is not in the RTAS stopped state) since the deconfigured processor cannot be started.

R1–7.2.6–3. The OF properties listed in Table 18, "OF Device Tree Properties," on page 115 must be in the **/rtas** device tree node prior to booting the OS.

Table 18. OF Device Tree Properties

name	value
"rtas-size"	integer size of RTAS private data area or zero if allocation is not required
"rtas-version"	An integer encoding of the RTAS interface version. This document describes version 1.
"rtas-event-scan-rate"	The rate, in calls per minute, at which rtas-event-scan should be called by the OS. See Section 7.3.3.1, "event-scan," on page 125.
"rtas-display-device"	The *phandle* of the device node used by the RTAS call, *display-character*
"rtas-error-log-max"	The maximum size of an extended error log.

R1–7.2.6–4. All RTAS functions listed as “Required” in Table 17, “RTAS Tokens for Functions,” on page 112 must be implemented in RTAS.

R1–7.2.6–5. For the Symmetric Multiprocessor option: The functions listed as “Required in SMP Platforms” in Table 17, “RTAS Tokens for Functions,” on page 112 must be implemented in RTAS.

Software Implementation Notes:

1. It is permitted for RTAS not to implement those optional functions that are not appropriate or not needed on a particular platform.
2. Vendors may introduce private RTAS calls of their own. If they do, the property names should be of the form **“vendor,property”** where **vendor** is a company name string as defined by OF. Future versions of this architecture will not choose RTAS property names that include a comma.

7.2.6.1 RTAS Device Tree Properties for Indicators and Sensors

Indicators and sensors may be static or dynamic. Each indicator or sensor type is identified by its token; a number which is associated with the functionality of the indicator or sensor. A specific indicator token is static in a particular platform if the number of indicators of that type do not change with Dynamic Reconfiguration (DR) operations, and dynamic otherwise. Certain sensors and indicators associated with DR[1] are static since they represent the base hardware, others are dynamic coming and going with extensions to the base hardware. Indices for DR indicators and sensors are obtained from the DRC index for the DRC connector. Information about static non-DR indicators and sensors (like indices and location codes) are specified in the OF device tree at boot time and do not change. Information about non-DR dynamic indicators and sensors, needs to be gathered via the *ibm,get-indices* RTAS call (see Section 7.3.16.21–4, “For the UUID option with the System Parameters option: For the UUID system parameter, the ibm,set-system-parameter RTAS call must always return a Status of -9002 (Setting not allowed/authorized).,” on page 234), and sensors, instead of being represented in the device tree.

Indicators and sensors within a platform generally have location codes associated with them. Location code information for static indicators and sensors, except DR indicators and sensors, are placed in the **“**<*vendor*>**,indicator-**<*token*>**”** and **“**<*vendor*>**,sensor-**<*token*>**”** properties, respectively, in the **/rtas** node, where “<vendor>” is defined in the column marked “<vendor>” in Table 40, “Defined Indicators,” on page 143 and Table 42, “Defined Sensors,” on page 147, respectively. Location code information for dynamic indicators and sensors, except DR indicators and sensors, for the most part come in via the *ibm,get-indices* call.

Information (index, location code) about a particular indicator or sensor token, except DR indicators and sensors, are in the **/rtas** node properties or are available via the *ibm,get-indices* call, but not both. When indices are provided via the **“rtas-sensors”** or **“rtas-indicators”** properties, it is expected that there exists a sensor/indicator for each index between 0 and *maxindex*. When indices are provided via the *ibm,get-indices* call, the indices may not be contiguous, and any of the indices between 0 and *maxindex* may be missing.

The formats for location codes are defined in Section 12.3, “Hardware Location Codes,” on page 327. For indicators and sensors, these location codes are for the location of the device being manipulated or measured, not the location of the specific controller or sensor. The location code for an abstracted indicator or sensor is a NULL string.

1.DR indicators include isolation-state (9001), DR indicator (9002), and allocation-state (9003). DR sensors include dr-entity-sense (9003). DR indicators and sensors are required to be there based on the DR entity being supported. Their indices are specified by the DR index for the DR entity. See Table 166, “get-sensor-state Defined Sensors for All DR Options,” on page 366 and Table 168, “set-indicator Defined Indicators for all DR Options,” on page 368 for more information.

7.2.6.1.1 Indicators

For static indicators, except DR indicators, OF provides for paired integers (*token maxindex*) for each indicator token under the property **"rtas-indicators"** in the **/rtas** node. With this information, the OS can determine which types of indicators, and the maximum number (*maxindex*) of each type, that the platform provides.

For static indicators, except DR indicators, the extension property, **"**<*vendor*>**,indicator-**<*token*>**"** (see Section B.6.3.1, "RTAS Node Properties," on page 690), provides an array of strings containing the FRU location codes associated with each indicator. See Section 12.3, "Hardware Location Codes," on page 327. Here, "<*vendor*>" corresponds to the "<vendor>" column of *Table 40, "Defined Indicators," on page 143* and "<*token*>" corresponds to the token of the **"rtas-indicators"** type. The index of a specific indicator token is used to index into the array up to *maxindex*.

Indices and location codes for dynamic indicators are obtained via the *ibm,get-indices* RTAS call and do not appear in the static properties in the **/rtas** node.

Indices for DR indicators 9001, 9002, and 9003 are obtained from the DRC index for the DRC connector. See Requirement R1–13.5.3.4–2.

R1–7.2.6.1.1–1. For all static indicators, except DR indicators 9001, 9002, and 9003, OF must provide the extension property, **"**<*vendor*>**,indicator-**<*token*>**"**, in the **/rtas** node, unless the indicator is part of an extension which has its own set of appropriate properties for the indicator, where "<*vendor*>" must be as defined in the column labeled "<vendor>" in Table 40, "Defined Indicators," on page 143 for the specific indicator token value.

R1–7.2.6.1.1–2. For all static indicators for which there is an associated **"**<*vendor*>**,indicator-**<*token*>**"** property and for which there is not a physical realization, the location code string must be NULL.

R1–7.2.6.1.1–3. Indices and location codes for any indicator token, except DR indicators 9001, 9002, and 9003, for which the number of such indicators in the platform may change dynamically, must be obtained via the *ibm,get-indices* RTAS call and the indicator token must not appear in the **"**<*vendor*>**,indicator-**<*token*>**"** or **"rtas-indicators"** in the **/rtas** node.

R1–7.2.6.1.1–4. The indicator token of 4 must not exist in a platform when a Error Log (token 9006) is implemented.

7.2.6.1.2 Sensors

For static sensors, except DR sensors, OF currently provides for paired integers (*token maxindex*) for each sensor token under the property **"rtas-sensors"** in the **/rtas** node. With this information, the OS can determine which types of sensors, and how many of each type, that the platform provides.

For static sensors, except DR sensors, the extension property, **"**<*vendor*>**,sensor-**<*token*>**"** (see Section B.6.3.1, "RTAS Node Properties," on page 690), provides an array of strings containing the FRU location codes associated with each sensor. See Section 12.3, "Hardware Location Codes," on page 327. Here, "<*vendor*>" corresponds to the "<vendor>" column of *Table 42, "Defined Sensors," on page 147* and "<*token*>" corresponds to the token in the **"rtas-sensors"** property. The index of a specific sensor is used to index into the array up to *maxindex*.

Indices and location codes for dynamic sensors, except DR sensors, are obtained via the *ibm,get-indices* RTAS call and do not appear in the static properties in the **/rtas** node.

Indices for DR sensors 9003 are obtained from the DRC index for the DRC connector. See Requirement R1–13.5.3.3–4.

R1–7.2.6.1.2–1. For all static sensors, except DR sensor 9003, OF must provide the extension property, **"**<*vendor*>**,sensor-**<*token*>**"**, in the **/rtas** node, unless the sensor is part of an extension which has its own

set of appropriate properties for the sensor, where "<*vendor*>" must be as defined in the column labeled "<vendor>" in Table 42, "Defined Sensors," on page 147 for the specific sensor token value.

R1–7.2.6.1.2–2. For all static sensors for which there is an associated **"**<*vendor*>**,sensor-**<*token*>**"** property and for which there is not a physical realization, the location code string must be NULL.

R1–7.2.6.1.2–3. Indices and location codes for any sensor token, except DR sensor 9003, for which the number of such sensors in the platform may change dynamically, must be obtained via the *ibm,get-indices* RTAS call and the sensor token must not appear in the **"**<*vendor*>**,sensor-**<*token*>**"** or **"rtas-sensors"** in the **/rtas** node.

R1–7.2.6.1.2–4. The following sensor tokens must not be implemented if the number of them may be changed by a DR operation (that is, they can only be used when static): 3, 9001, and 9002.

7.2.7 Calling Mechanism and Conventions

RTAS is called through a mechanism similar to the OF client interface service. An argument buffer is constructed which describes the desired RTAS call. This description includes an indication of the RTAS call that is being invoked, the number and value of the input parameters, the number of result values, and space for each of the result values.

R1–7.2.7–1. In order to make an RTAS call, the OS must construct an argument call buffer aligned on an eight byte boundary in physically contiguous real memory as described by Table 19, "RTAS Argument Call Buffer," on page 118.

Table 19. RTAS Argument Call Buffer

Cell Number	Use
1	Token Specifying which RTAS Call
2	Number of Input Parameters
3	Number of Output Parameters
4	First Input Parameter
...	Other Input Parameters
4 + Number of Inputs -1	Last Input Parameter
4 + Number of Inputs	First Output Parameter
...	Other Output Parameters
4 + Number of Inputs + Number of Outputs -1	Last Output Parameter

R1–7.2.7–2. If the system is a 32-bit system, or if RTAS was instantiated by **instantiate-rtas**, then all cells in the RTAS argument buffer must be 32-bit sign extended values that are aligned to 4 byte boundaries.

R1–7.2.7–3. If the system is a 64 bit system and if RTAS was instantiated by **instantiate-rtas-64**, then all cells in the RTAS argument buffer must be 64-bit sign extended values that are aligned to 8 byte boundaries.

R1–7.2.7–4. RTAS functions must be invoked by branching to the *rtas-call* address which is returned by the **`instantiate-rtas`** or **`instantiate-rtas-64`** OF method (see Table 16, “instantiate-rtas or instantiate-rtas-64 Argument Call Buffer,” on page 112).

R1–7.2.7–5. Register R3 must contain the argument buffer’s real address when *rtas-call* is invoked.

R1–7.2.7–6. Register R4 must contain the real address of the RTAS private data area when *rtas-call* is invoked (see Requirement R1–7.2.4–1).

R1–7.2.7–7. The Link Register must contain the return address when *rtas-call* is invoked.

Software Implementation Notes:

1. RTAS is not required to perform sanity checking of its input parameters. Using invalid values for any parameter in an RTAS argument buffer gives undefined results. However, when such checks are made, the appropriate return code for invalid parameters is -3.
2. The token that specifies the RTAS call is obtained by looking up the desired call from the **`/rtas`** node of the OF device tree.
3. The OS must be aware that the effective address range for RTAS is 4 GB when instantiated in 32-bit mode and the OS should not pass RTAS addresses or blocks of data which might fall outside of this range.

7.2.8 Return Codes

R1–7.2.8–1. The first output value of all the RTAS functions must be a *Status* word which denotes the result of the call. The *Status* word takes on one of the values in Table 20, “RTAS Status Word Values,” on page 119. Non-negative values indicate success.

Table 20. RTAS *Status* Word Values

Values	*Status* Word Meanings
0	RTAS function call succeeded.
-1	RTAS function call encountered a hardware error.
-2	A necessary hardware device was busy, and the requested function could not be performed. The operation should be retried at a later time.
-3	Parameter Error. In some cases, specific parameter errors are enumerated. However, other parameter errors may be reported using this return code in addition to those enumerated.
-7	Unexpected state change.
9000-9899	Reserved for vendor specific success codes.
990x	Extended delay - where x is a number in the range of 0-5
-9000	Multi-level isolation error (see Section 13.7.4.1.3, “Isolation of PHBs and Slots,” on page 381).
-9004 - (-9999)	Reserved for vendor specific error codes.
Additional Negative Numbers	An error was encountered. The meaning of this error is specific to the RTAS function that was invoked.

Table 20. RTAS *Status* Word Values

Values	*Status* Word Meanings
Additional Positive Numbers	The function succeeded. The meaning of the *Status* word is specific to the RTAS function that was invoked.

Table 20, “RTAS Status Word Values,” on page 119 indicates a summation of all possible *Status* word values. A given RTAS function cannot yield all of the possible *Status* words. For the specific *Status* words which apply to a specific RTAS function, see the semantics for that function.

Software Implementation Notes:

1. A return code of -2 or 990x may either mean that the operation was initiated but not completed, or may mean that the operation was not initiated at all.
2. When the extended delay (990x) is returned, it is suggested that software delay for 10 raised to the x milliseconds, where x is the last digit of the 990x return code, before calling the function again.

7.3 RTAS Call Function Definition

This section specifies the semantics of all the RTAS calls. It specifies the RTAS function name, the contents of its argument call buffer (its token, input parameters, and output values) and semantics.

7.3.1 NVRAM Access Functions

This architecture requires an area of non-volatile memory (NVRAM) to hold OF options, RTAS information, machine configuration state, OS state, diagnostic logs, etc. The type and size of NVRAM is specified in the OF device tree. The format of NVRAM is detailed in Chapter 8, “Non-Volatile Memory,” on page 265.

In order to give the OS the ability to access NVRAM on different platforms that may use different implementations or locations for NVRAM, a layer of abstraction is provided to the OS. The functions in this section provide an interface for reading and writing NVRAM with byte level operations with no boundary requirements.

7.3.1.1 *nvram-fetch*

The RTAS function *nvram-fetch* copies data from a given offset in NVRAM into the user specified buffer.

R1–7.3.1.1–1. RTAS must implement an *nvram-fetch* function that returns data from NVRAM using the argument call buffer defined by Table 21, “nvram-fetch Argument Call Buffer,” on page 121.

Table 21. *nvram-fetch* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *nvram-fetch*
	Number Inputs	3
	Number Outputs	2
	Index	Byte offset in NVRAM
	Buffer	Real address of data buffer
	Length	Size of data buffer (in bytes)
Out	*Status*	0: Success -1: Hardware Error -3: Parameter out of range
	Num	Number of bytes successfully copied

7.3.1.2 *nvram-store*

The RTAS function *nvram-store* copies data from the user specified buffer to a given offset in NVRAM.

R1–7.3.1.2–1. RTAS must implement an *nvram-store* function that stores data in NVRAM using the argument call buffer defined by Table 22, “nvram-store Argument Call Buffer,” on page 121.

Table 22. *nvram-store* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *nvram-store*
	Number Inputs	3
	Number Outputs	2
	Index	Byte number in NVRAM
	Buffer	Real address of data buffer
	Length	Size of data buffer (in bytes)
Out	*Status*	0: Success -1: Hardware Error -3: Parameter out of range
	Num	Number of bytes successfully copied

R1–7.3.1.2–2. If the *nvram-store* operation succeeded, the contents of NVRAM must have been updated to the user specified values. The contents of NVRAM are undefined if the RTAS call failed.

Platform Implementation Note: The platform may keep the NVRAM data cached in volatile memory as long as the cache is implemented as a store-through cache and not a store-in cache. That is, changed data is written to

NVRAM as soon as possible. Return from the *nvram-store* call with a "success" *Status* is permissible after placing the data into a store-through cache and prior to the actual writing to the NVRAM.

R1–7.3.1.2–3. The caller of the *nvram-store* RTAS call must maintain the NVRAM partitions as specified in Chapter 8, "Non-Volatile Memory," on page 265.

7.3.2 Time of Day

The minimum system requirements include a non-volatile real time clock which maintains the time of day even if power to the machine is removed. Minimum requirements for this clock are described in Requirement R1–2.9–7.

7.3.2.1 Time of Day Inputs/Outputs

The OS maintains the clock in UTC. This allows the OS and diagnostics to co-exist with each other and provide uniform handling of time.

R1–7.3.2.1–1. The date and time inputs and outputs to the RTAS time of day function calls are specified with the year as the actual value (for example, 1995), the month as a value in the range 1-12, the day as a value in the range 1-31, the hour as a value in the range 0-23, the minute as a value in the range 0-59, and the second as a value in the range 0-59. The date must also be a valid date according to common usage: the day range being restricted for certain months, month 2 having 29 days in leap years, etc.

R1–7.3.2.1–2. OSs must account for local time, for daylight savings time when and where appropriate, and for leap seconds.

R1–7.3.2.1–3. RTAS must account for leap years.

7.3.2.2 *get-time-of-day*

R1–7.3.2.2–1. RTAS must implement a *get-time-of-day* call using the argument call buffer defined by Table 23, "get-time-of-day Argument Call Buffer," on page 122.

Table 23. *get-time-of-day* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *get-time-of-day*
	Number Inputs	0
	Number Outputs	8

Table 23. *get-time-of-day* Argument Call Buffer ***(Continued)***

Parameter Type	Name	Values
Out	*Status*	990x: Extended Delay where x is a number 0-5 (see text below) 0: Success -1: Hardware Error -2: Clock Busy, Try again later
	Year	Year
	Month	1-12
	Day	1-31
	Hour	0-23
	Minute	0-59
	Second	0-59
	Nanoseconds	0-999999999

Software Implementation Note: When the 990x *Status* is returned, it is suggested that software delay for 10 raised to the x milliseconds (where x is the last digit of the 990x return code), before calling again. However, software may issue the call again either earlier or later than this.

R1–7.3.2.2–2. RTAS must read the current time and set the output values to the best resolution provided by the platform.

7.3.2.3 *set-time-of-day*

R1–7.3.2.3–1. RTAS must implement a *set-time-of-day* call using the argument call buffer defined by Table 24, “set-time-of-day Argument Call Buffer,” on page 123.

Table 24. *set-time-of-day* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *set-time-of-day*
	Number Inputs	7
	Number Outputs	1
	Year	Year
	Month	1-12
	Day	1-31
	Hour	0-23
	Minute	0-59
	Second	0-59
	Nanosecond	0-999999999

Table 24. *set-time-of-day* Argument Call Buffer ***(Continued)***

Parameter Type	Name	Values
Out	*Status*	990x: Extended Delay where x is a number 0-5 (see text below) 0: Success -1: Hardware Error -3: Parameter Error

Software Implementation Note: When the 990x *Status* is returned, it is suggested that software delay for 10 raised to the x milliseconds (where x is the last digit of the 990x return code), before calling again. However, software may issue the call again either earlier or later than this.

R1–7.3.2.3–2. RTAS must set the time of day to the best resolution provided by the platform.

R1–7.3.2.3–3. RTAS must return a *Status* of -3 (Parameter Error) to the *set-time-of-day* RTAS call when the specified date is outside the range supported by the platform.

Software Implementation Note: The OS maintains the clock in UTC. This allows the OS and diagnostics to co-exist with each other and provide uniform handling of time. Refer to Requirement R1–2.9–7 for further details on the time of day clock.

7.3.2.4 *set-time-for-power-on*

Some platforms provide the ability to set a time to cause the platform power on. The *set-time-for-power-on* call provides the interface to the OS for setting this timer.

R1–7.3.2.4–1. RTAS must implement the *set-time-for-power-on* call using the argument call buffer defined by Table 25, “set-time-for-power-on Argument Call Buffer,” on page 124.

Table 25. *set-time-for-power-on* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *set-time-for-power-on*
	Number Inputs	7
	Number Outputs	1
	Year	Year
	Month	1-12
	Day	1-31
	Hour	0-23
	Minute	0-59
	Second	0-59
	Nanosecond	0-999999999
Out	*Status*	990x: Extended Delay where x is a number 0-5 (see text below) 0: Success -1: Hardware Error -2: Clock Busy, Try again later -3: Parameter Error

Software Implementation Note: When the 990x *Status* is returned, it is suggested that software delay for 10 raised to the x milliseconds (where x is the last digit of the 990x return code), before calling again. However, software may issue the call again either earlier or later than this.

R1–7.3.2.4–2. Hardware must support power on times of up to four weeks into the future, at a minimum.

R1–7.3.2.4–3. RTAS must schedule the time for power on as close as it can approach to the desired time.

Software Implementation Note: Hardware limitations on the duration of the power-on timer may result in power-on sooner than requested by software. If present in the `/rtas` node, the OF property `"power-on-max-latency"` gives in days the maximum power-on duration capability of the hardware. If the property is not present, software should expect the default of a maximum of 28 days. A "day" is defined as 24 hour increments from the current time.

R1–7.3.2.4–4. If the system is in a powered down state at the time scheduled by *set-time-for-power-on* (within the accuracy of the clock), then power must be reapplied and the system must go through its power on sequence.

R1–7.3.2.4–5. RTAS must return a *Status* of -3 (Parameter Error) to the *set-time-for-power-on* RTAS call when the specified date is outside the range supported by the platform (such as before current TOD).

7.3.3 Error and Event Reporting

The error and event reporting RTAS calls are designed to provide an abstract interface into hardware registers in the system that may contain correctable or non-correctable errors and to provide an abstract interface to certain platform events that may be of interest to the OS. Such errors and events may be detected either by a periodic scan or by an exception trap.

These functions are not intended to replace the normal error handling in the OS. Rather, they enhance the OS's abilities by providing an abstract interface to check for, report, and recover from errors or events on the platform that are not necessarily known to the OS.

The OS uses the error and event RTAS calls in two distinct ways:

1. Periodically, the OS calls *event-scan* to have the system firmware check for any errors or events that have occurred.
2. Whenever the OS receives an interrupt or exception that it cannot fully process, it calls *check-exception.*

The first case covers all errors and events that do not signal their occurrence with an interrupt or exception. The second case covers those errors and events that do signal with an interrupt or exception. It is platform dependent whether any specific error or event causes an interrupt on that platform.

R1–7.3.3–1. RTAS must return the event generated by a particular interrupt or event source by either *check-exception* or *event-scan*, but not both.

R1–7.3.3–2. *check-exception* and *event-scan*, on a 64-bit capable platform, must be able to handle platform resources that are accessed using 64-bit addresses when instantiated in 32-bit mode.

7.3.3.1 *event-scan*

R1–7.3.3.1–1. RTAS must implement an *event-scan* call using the argument call buffer defined by Table 26, "event-scan Argument Call Buffer," on page 126.

Table 26. *event-scan* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *event-scan*
	Number Inputs	4
	Number Outputs	1
	Event Mask	Mask of event classes to process
	Critical	Indicates whether this call is required to complete quickly
	Buffer	Real address of error log
	Length	Length of error log buffer
Out	*Status*	1: No Errors Found 0: New Error Log returned -1: Hardware Error

R1–7.3.3.1–2. The *event-scan* call must fill in the error log with a single error log formatted as specified in Section 10.3.2, “RTAS Error/Event Return Format,” on page 289. If necessary, the data placed into the error log must be truncated to *length* bytes.

R1–7.3.3.1–3. RTAS must only check for errors or events that are within the classes defined by the *Event mask*. *Event mask* is a bit mask of error and event classes. Refer to Table 134, “Error and Event Classes with RTAS Function Call Mask,” on page 282 for the definition of the bit positions.

R1–7.3.3.1–4. If *Critical* is non-zero, then RTAS must perform only those operations that are required for continued operation. No extended error information is returned.

R1–7.3.3.1–5. The *event-scan* call must return the first found error or event and clear that error or event so it is only reported once.

R1–7.3.3.1–6. The OS must continue to call *event-scan* while a *Status* of “New Error Log returned” is returned.

R1–7.3.3.1–7. The *event-scan* call must be made at least **`"rtas-event-scan-rate"`** times per minute for each error and event class and must have the *Critical* parameter equal to 0 for this periodic call.

R1–7.3.3.1–8. The platform must not return more than two error logs during the first sequence of *event-scan* RTAS calls after boot of an OS image, and must not return more than one error log to that OS image during any sequence of *event-scan* RTAS calls after the first time a non-zero Status is returned.

Software Implementation Notes:

1. In a multiprocessor system, each processor should call *event-scan* periodically, not always the same one. The *event-scan* function needs to be called a total of **`"rtas-event-scan-rate"`** times a minute.
2. The maximum size of the error log is specified in the OF device tree as the **`"rtas-error-log-max"`** property of the **`/rtas`** node.
3. This call does not log the error in NVRAM. It returns the error log to the OS. It is the responsibility of the OS to take appropriate action.
4. For best system performance, the requested **`"rtas-event-scan-rate"`** should be as low as possible, and as a goal should not exceed 120 scans per minute. Maximum system performance is obtained when no scans are required.

7.3.3.2 *check-exception*

R1–7.3.3.2–1. RTAS must implement a *check-exception* call using the argument call buffer defined by Table 27, “check-exception Argument Call Buffer,” on page 127.

Table 27. *check-exception* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *check-exception*
	Number Inputs	6 (without Extended Information) 7 (with Extended Information)
	Number Outputs	1
	Vector Offset	The vector offset for the exception. See *Power ISA* [1].
	Additional Information	Information which RTAS may need to determine the cause of the exception, but which may be unavailable to it in hardware registers. See Table 28, “Additional Information Provided to check-exception call,” on page 127 for details.
	Event Mask	Mask of event classes to process
	Critical	Indicates whether this call is required to complete quickly
	Buffer	Real address of error log
	Length	Length of error log
	Extended Information	See Requirement R1–7.3.3.2–2
Out	*Status*	1: No Errors Found 0: New Error Log returned -1: Hardware Error

R1–7.3.3.2–2. The OS must provide the value specified in Table 28, “Additional Information Provided to check-exception call,” on page 127 in the *Additional Information* parameter in the call to *check-exception*, with the *Number Inputs* parameter set to 6. If the value (e.g., SRR1) is too large to fit in this cell, the lower 32-bits must be provided here, the upper 32-bits provided in the *Extended Information* parameter, and the *Number Inputs* parameter set to 7.

Table 28. Additional Information Provided to *check-exception* call

Source of Interrupt	Value of “Additional Information” Variable
External Interrupt	Interrupt number
Machine check exception	Value of register SRR1 at entry to machine check handler
System Reset exception	Value of register SRR1 at entry to system reset handler
Other exception	Value of register SRR1 at entry to exception handler

R1–7.3.3.2–3. The *check-exception* call must fill in the error log with a single error log formatted as specified in Section 10.3.2, “RTAS Error/Event Return Format,” on page 289. The data in the error log must be truncated to *length* bytes.

R1–7.3.3.2–4. If *Critical* is non-zero, then RTAS must perform only those operations that are required for continued operation. No extended error information is returned.

R1–7.3.3.2–5. The *check-exception* call must return the first found error or event and clear that error or event so it is only reported once.

R1–7.3.3.2–6. RTAS must only check for errors or events that are within the classes defined by the *Event mask. Event mask* is a bit mask of error and event classes. Refer to Table 134, "Error and Event Classes with RTAS Function Call Mask," on page 282 for the definition of the bit positions.

Software Implementation Notes:

1. All OS reserved exception handlers should call *check-exception* to process any errors that are unknown to the OS.
2. The *interrupt number* for external device interrupts is provided in the OF device tree as specified in Appendix B, "LoPAPR Binding," on page 661.
3. Software, with knowledge of the class of event it seeks, matches the data in the *Vector Offset*, *Additional Information*, and *Extended Information* with the *Event Mask* such that ambiguity does not result.

7.3.3.3 *rtas-last-error*

R1–7.3.3.3–1. RTAS must implement an *rtas-last-error* call using the argument call buffer defined in Table 29, "rtas-last-error Argument Call Buffer," on page 128.

Table 29. *rtas-last-error* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *rtas-last-error*
	Number Inputs	2
	Number Outputs	1
	Buffer	Real address of error log
	Length	Length of error log buffer
Out	*Status*	1: No Errors Found 0: New Error Log Returned -1: Hardware Error (cannot create log)

R1–7.3.3.3–2. The *rtas-last-error* call must fill in the error log with a single error log formatted as specified in Section 10.3.2, "RTAS Error/Event Return Format," on page 289. If necessary, the data placed into the error log must be truncated to 'length" bytes.

R1–7.3.3.3–3. RTAS must only check for hardware errors that occurred during a prior call to some other RTAS function, resulting in a -1 (Hardware Error) return *Status*.

Software Note: This function is intended to provide the OS with more detailed failure information after an RTAS call returns with a -1 (Hardware Error) *Status*, and should not be called except for this purpose. If *rtas-last-error* itself

returns a -1 *Status*, then it could not create the error log data because of a further error, and the OS should not try to call it again.

7.3.3.4 Platform Dump Option

The architectural intent of the Platform Dump option is to allow a mechanism for the platform to communicate a variety of dump data used to debug problems within the platform firmware or hardware.

7.3.3.4.1 *ibm,platform-dump*

This RTAS call is used to transfer dump data from the platform to the OS. It is expected that this routine will have to be called several times to complete the transfer of the diagnostic dump data. It is also anticipated that multiple dumps could be in the process of completion at the same time. Individual dumps are identified by a dump tag passed by the OS. The OS may interleave calls to *ibm,platform-dump* with different RTAS calls. Other standard RTAS locking rules apply (for example, only one processor may call RTAS at a time).

The OS only makes the *ibm,platform-dump* RTAS call when an event scan returns an error log with an Event Type of "Dump Notification" as described in Version 6 or later of the RTAS General Extended Error Log Format.

R1–7.3.3.4.1–1. For the Platform Dump option: The RTAS function *ibm,platform-dump* must be implemented and must implement the argument call buffer as defined by Table 30, "ibm,platform-dump Argument Call Buffer," on page 129.

Table 30. *ibm,platform-dump* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,platform-dump*
	Number of Inputs	6
	Number of Outputs	5
	Dump_Tag_Hi	Most-significant 32 bits of a Dump_Tag representing an id of the dump being processed
	Dump_Tag_Lo	Least-significant 32 bits of a Dump_Tag representing an id of the dump being processed
	Sequence_Hi	Most-significant 32 bits of the Sequence, a value indicating what portion of a dump to be returned by the call. Sequence of 0 returns the beginning of the Dump. The value in all subsequent call as needed, should be set to the value of the Next_Sequence returned from each previous call.
	Sequence_Lo	Least-significant 32 bits of the Sequence
	Buffer	Address of dump buffer (NULL indicates completion of processing)
	Length	Length of the buffer in bytes (min. 1024)

Table 30. *ibm,platform-dump* Argument Call Buffer ***(Continued)***

Parameter Type	Name	Values
Out	*Status*	-1: Hardware error -2: Busy, try again later -9002: Not Authorized 0: Dump complete 1: Continue dump 990x: Extended Delay where x is a number 0-5
	Next_Sequence_Hi	Most-significant 32 bits of the Next_Sequence value indicating the portion of the dump to be retrieved on the next call if needed. (If *Status* is returned as 0, then the dump is complete and there is no next call required. The value of Next_Sequence in this case is undefined.)
	Next_Sequence_Lo	Least-significant 32 bits of the Next_Sequence value
	Bytes_Returned_Hi	Most-significant 32 bits of the Bytes_Returned value indicating the number of valid bytes returned in the Buffer
	Bytes_Returned_Lo	Least-significant 32 bits of the Bytes_Returned value indicating the number of valid bytes returned in the Buffer

Software Implementation Note: When the 990x *Status* is returned, it is suggested that software delay for 10 raised to the x milliseconds (where x is the last digit of the 990x return code), before calling *ibm,platform-dump* again. However, software may issue the *ibm,platform-dump* call again either earlier or later than this.

R1–7.3.3.4.1–2. For the Platform Dump option: On the first call to *ibm,platform-dump* of a platform dump sequence for a given Dump_Tag, the Sequence value must be initialized to zero and on subsequent calls for the same tag, the Dump_Sequence must be set to Next_Sequence of the previous call made with the same Dump_Tag or else set to zero to restart the entire dump sequence.

R1–7.3.3.4.1–3. For the Platform Dump option: The dump tag passed to any call to ibm,platform-dump must be a value specified by the platform and communicated to the OS by an *event-scan* error log entry.

R1–7.3.3.4.1–4. For the Platform Dump option: Once a *Status* of 0 (Dump complete) or -1 (Hardware error" is returned for the *ibm,platform-dump* call with a particular dump tag, the dump is considered complete from a platform standpoint, but for the "Dump complete" case the OS must signal to the platform that the processing of the dump has been completed by a final call for the Dump_Tag with the Buffer address set to NULL.

R1–7.3.3.4.1–5. For the Platform Dump option: If at any time a partition receives a -9002, Not Authorized, return code for an *ibm,platform-dump* RTAS, the partition must cease attempting to acquire the dump information it was in process of acquiring and discard any portion already acquired.

Programming Note: It is expected that a platform generally only transmits a dump to a single partition. However, the above requirement makes provision for the platform abandoning the transmission of a dump to a partition after it has been initiated, presumably to re-initiate transmission to a different partition or to a Hardware Management Console (HMC).

R1–7.3.3.4.1–6. For the Platform Dump option: The contents of dump information returned through the sequence of calls to *ibm,platform-dump*, must follow a dump directory structure as defined in Section 7.3.3.4.2, "Platform Dump Directory Structure," on page 131.

R1–7.3.3.4.1–7. For the Platform Dump option: Collectively the dump data returned from a sequence of *ibm,platform-dump* calls for a given Dump_tag must consist of one dump file directory entry as described in Table 31, "Platform Dump File Directory Entry Format," on page 131 followed by one or more dump section directory entries as described in Table 32, "Dump Section Directory Entry Format," on page 132 followed by a dump data section for each dump section directory entry earlier included.

Programming Notes:

1. As required in Section 7.3.16.10, “Platform Dump Max Size Parameter,” on page 224, the OS can determine the maximum size of a copy of each dump that can be returned by issuing an *ibm,get-system-parameter* for the platform-dump-max-size. In addition, in the case of any change in the value of this parameter, the platform may generate a Platform Event Log entry announcing the change in the maximum size, and specifying the new size in the IO Events Section. This entry, when generated, is then returned by the *event-scan* RTAS call.

2. The Dump_Tag is taken from the Dump Locator Section of the Platform Error/Event Log Format, Version 6 or later. Specifically, Dump_Tag_Hi is composed of the 8 bit Dump Type as found in the Dump Locator Section, padded with 24 bits on the left to make a 32 bit quantity. The Dump_Tag_Lo is the Dump ID found in the Dump Locator Section of the Error log entry.

3. If the *ibm,platform-dump* RTAS routine returns with the *Status* of 1 (Continue dump), the transfer is proceeding but had to be suspended to maintain the short execution time requirement of RTAS routines or because more data was available than the Buffer could contain.

4. The Bytes_Returned value indicates how many bytes of dump data (if any) were returned on a call and OS must be prepared to handle the case of no bytes returned. When Continue dump *Status* (1) is returned, this indicates that there is more dump data available then was returned in the buffer. A subsequent call with the same Dump_tag and the Sequence value being set to the Next_Sequence returned from the previous call returns additional dump data.

5. When a dump has been successfully transmitted, the *Status* of 0 (Dump complete) is returned. If there is a hardware error preventing a dump from being successfully transmitted, as *Status* of -1 (Hardware error) is returned. In either case, the Dump sequence is completed. It should be noted that the final Next_Sequence value returned is undefined. After the sequence is completed, the OS should make one final call for the given Dump_Tag using a NULL buffer pointer. (The value of the Sequence parameter for this call is undefined although it is acceptable for the platform to make the value equal to the last Next_Sequence value returned.) This call tells the platform that the OS has completed processing of the dump and will not attempt to restart the sequence.

6. If the platform used system memory to hold dump data, the platform at this point is permitted to free the associated logical memory blocks (LMBs) reserved for the dump. Successful return from the *ibm,platform-dump* RTAS call with a NULL buffer pointer indicates to the OS that one or more logical memory blocks (LMBs) may now be acquired by the OS. A get-sensor-state RTAS call for these LMBs returns with a state of “DR entity available for recovery (4)” after the successful return from this *ibm,platform-dump* RTAS call.

7. If a platform does not receive the NULL buffer pointer call dump for a given Dump_Tag but subsequently boots the partition, the platform may report the presence of the dump again on an e*vent-scan* after the boot.

7.3.3.4.2 Platform Dump Directory Structure

The entire dump contents returned over a sequence of *ibm,platform-dump* RTAS calls for a given Dump_Tag follows a directory/data structure as illustrated in Table 31, “Platform Dump File Directory Entry Format,” on page 131 and Table 32, “Dump Section Directory Entry Format,” on page 132 where a dump consists of one File Directory Entry, one or more Section Directory Entries and one data section for each Section Directory entry.

Table 31. Platform Dump File Directory Entry Format

Field Name	Length	Values	Discussion
Entry Header	8 Bytes	“FILE”	Identifies the type of entry that follows. The value is ASCII consisting of the characters “FILE” and 4 ASCII blanks.
Entry Length	2 Bytes	Number of bytes of the entire file directory entry	This length includes the Entry Header and Entry Length fields.

Table 31. Platform Dump File Directory Entry Format

Field Name	Length	Values	Discussion
Reserved	6 Bytes		
Flags	4 Bytes	See Table 33, “Dump File Format Directory Options,” on page 133.	
Entry Type	2 Bytes	0x0001	0x0001signifies a file entry.
Prefix Length	2 Bytes	Number of bytes of the Dump File Base Name that is considered to be a prefix.	
Dump File Base Name	Length in bytes computed as “Entry length” - 24, but not to exceed 46 characters including the ASCII NULL string termination.	NULL terminated ASCII String consisting of ASCII characters in the ranges of a-z, A-Z, 0-9, and the ASCII “.”	Gives a base name for the dump file to be created from the dump data. This base name is composed of a prefix followed by additional data (e.g. dumptype.serialnumber.dumpID.timestamp where dumptype.serialnumber is the prefix)

Table 32. Dump Section Directory Entry Format

Field Name	Length	Values	Discussion
Entry Header	8 Bytes	“SECTION”	Identifies the type of entry that follows. The value is ASCII consisting of the characters “SECTION” and 1 ASCII blank.
Entry Length	2 Bytes	Number of bytes of the entire section directory entry	This length includes the Entry Header and Entry Length fields.
Priority	2 Bytes	Unsigned integer	See programming note after Table 33, “Dump File Format Directory Options,” on page 133.
Reserved	4 Bytes		
Flags	4 Bytes	See Table 33, “Dump File Format Directory Options,” on page 133.	
Entry Type	2 Bytes	0x0002	0x0002 signifies a section entry.
Reserved	2 Bytes		
Section Length	8 Bytes	Length in bytes of the section of the dump that this entry is the directory for	
Section Name	Length in bytes computed as “Entry length” - 32, but not to exceed 46 characters including the ASCII NULL string termination.	NULL terminated ASCII String.	Gives a name to the dump section for which this entry is a directory.

The two previous tables refer to a set of flags used to describe information related to a dump section. The options are stored in a single 32 bit value which is the bit-wise OR'ing of each option value defined in Table 33, “Dump File Format Directory Options,” on page 133.

Table 33. Dump File Format Directory Options

Name	Bit Position(s) of Option	Definition	Discussion
last_flag	0x00000001	Binary value set to 1 if the last directory entry.	Flag is never set for the File Directory entry since at least one Section Directory entry follows.
not_transmitted	0x00000002	If set to 1, indicates that the data for the block has not been transmitted during some process of dump transfer.	Platform always sets this value to 0. The bit may be set to 1 by applications transmitting a dump. See Software implementation note item 2: in this section below.
Reserved	All but bit positions shown above	All other values reserved	

Software Implementation Notes:

1. Platforms supporting the *ibm,platform-dump* call may have several unique dump types. All dumps of the same type on a partition have the same "prefix" to the name of the dump file as indicated in the dump file directory entry in an error log.

2. The priority in the priority field of the section directory entries allow an application transmitting a dump to a remote support center to decide what sections of data to transmit when the connection bandwidth is limited. Zero is the highest priority. All sections at the same priority shall be transmitted if any at that priority are transmitted. It is intended that all directory entries be transmitted with the section length set to zero and their not_transmitted Dump File Format Directory Options flag set to a 1 if the section data cannot be transmitted.

7.3.4 PCI Configuration Space

Device drivers and system software need access to PCI configuration space. Chapter 3, "Address Map," on page 59 defines system address spaces for PCI memory and PCI I/O spaces. It does not define an address space for PCI configuration. Different PCI bridges may implement the mechanisms for accessing PCI configuration space in different ways. The RTAS calls in this section provide an abstract way of reading and writing PCI configuration spaces.

The PCI access functions take a *config_addr* input parameter which is similar to the Type 1 PCI configuration space address. For conventional PCI and PCI-X Mode 1, this address is a 24-bit quantity composed of bus, device, function, and register numbers. This allows the configuration of up to 256 buses (including sub-bridges), 32 IOAs per bus, 8 functions per IOA, and 256 bytes of register space per function. PCI-X Mode 2 and PCI Express define an extended configuration space with an additional 4-bit quantity which specifies an extended register number allowing for 4096 bytes of register space per function. Refer to the *PCI-X Protocol Addendum to the PCI Local Bus Specification* [21] or the *PCI Express Base Specification* [22] for more details. The *config_addr* for an IOA is derived from the OF device tree, and is defined in Table 34, "Config_addr Definition," on page 134.

The *ibm,read-pci-config* and *ibm,write-pci-config* RTAS calls allow for the specification of the PHB Bus Unit ID, and therefore allow for up to 256 unique *config_addr* bus numbers per PHB. Note that for each pci connector, there may be multiple PCI bus numbers, because plug-in PCI cards may contain PCI to PCI bridges, which create other PCI buses.

The *PCI Local Bus Specification* requires that unimplemented or reserved register space read as 0's, and that reads of the Vendor ID register of IOAs or functions which aren't present should be unambiguously reported (reading 0xFFFF is sufficient). Writes to unimplemented or reserved register space are specified as no-ops. Writes to IOAs or functions which aren't present are undefined. These operations are undefined if a bus is specified which doesn't exist.

R1–7.3.4–1. For the RTAS PCI configuration space and EEH functions where the parameter *config_addr* is requested as input, the *config_addr* parameter must be as specified by the hi cell of the physical address in

Open Firmware Working Group proposal number 516 Ver 1.8 (see Table 34, “Config_addr Definition,” on page 134), with the upper register address bits added for PCI-X Mode 2 and PCI Express, in order to access past the first 256 bytes of configuration space.

Table 34. *Config_addr* Definition

Bit	Definition
0:3	Upper bits of the Register Number, when applicable, otherwise 0. Set to 0 when the PCI extended configuration space is not available, due to lack of support somewhere from the PHB to the IOA. When a value of this field can be something other than 0, the **“ibm,pci-config-space-type”** property will exist in the IOA's node with a value indicating that the extended space is supported.
4:7	Reserved (set to 0)
8:15	Bus Number
16:20	Device Number
21:23	Function Number, when applicable, otherwise 0
24:31	Lower bits of the Register Number, when applicable, otherwise 0

R1–7.3.4–2. All RTAS PCI Read/Write functions must follow the appropriate PCI specification.

R1–7.3.4–3. RTAS must follow the rules of Table 10, “Big-Endian Mode Load and Store Programming Considerations,” on page 79 when accessing PCI configuration space.

Software Implementation Notes:

1. Since PCI Configuration space is defined to be Little-Endian, RTAS accesses this area using the byte-reversed forms of the *Load* and *Store* instructions. In this fashion, the values passed are defined Big-Endian.

2. Prior to accessing the extended configuration address space of PCI-X Mode 2 and PCI Express devices, an IOA device driver is responsible for checking if the **“ibm,pci-config-space-type”** property (see Section B.6.5.1.1, “PCI Host Bridge Properties,” on page 701) of the IOA's node exists and is set to a non-zero value.

7.3.4.1 *ibm,read-pci-config*

R1–7.3.4.1–1. For Platforms which may have greater than 256 PCI Buses: RTAS must implement an *ibm,read-pci-config* call using the argument call buffer defined by Table 35, “ibm,read-pci-config Argument Call Buffer,” on page 135.

Table 35. *ibm,read-pci-config* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,read-pci-config*
	Number Inputs	4
	Number Outputs	2
	Config_addr	Configuration Space Address
	PHB_Unit_ID_Hi	Represents the most-significant 32-bits of the Unit ID of the PHB that corresponds to the *config_addr*
	PHB_Unit_ID_Low	Represents the least-significant 32-bits of the Unit ID of the PHB that corresponds to the *config_addr*
	Size	Size of Configuration Cycle in bytes, value can be 1, 2, or 4
Out	*Status*	0: Success -1: Hardware Error -3: Parameter Error
	Value	Value Read from the location specified by the PHB Unit ID and *config_addr*

R1–7.3.4.1–2. The *ibm,read-pci-config* call must return the value from the configuration register which is at the location specified by the PHB Unit ID and *config_addr* in PCI configuration space.

R1–7.3.4.1–3. The *ibm,read-pci-config* call must perform a 1-byte, 2-byte, or 4-byte configuration space read depending on the value of the *size* input argument.

R1–7.3.4.1–4. The *config_addr* must be aligned to a 2-byte boundary if *size* is 2 and to a 4-byte boundary if *size* is 4.

R1–7.3.4.1–5. The *ibm,read-pci-config* call of IOAs or functions which are not present or which are not available to the caller must return *Success* with all ones as the output *value*.

7.3.4.2 *ibm,write-pci-config*

R1–7.3.4.2–1. For Platforms which may have greater than 256 PCI Buses: RTAS must implement an *ibm,write-pci-config* call using the argument call buffer defined by Table 36, “ibm,write-pci-config Argument Call Buffer,” on page 136.

Table 36. *ibm,write-pci-config* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,write-pci-config*
	Number Inputs	5
	Number Outputs	1
	Config_addr	Configuration Space Address
	PHB_Unit_ID_Hi	Represents the most-significant 32-bits of the Unit ID of the PHB that corresponds to the *config_addr*
	PHB_Unit_ID_Low	Represents the least-significant 32-bits of the Unit ID of the PHB that corresponds to the *config_addr*
	Size	Size of Configuration Cycle in bytes, can be 1, 2, or 4
	Value	Value to be written to the location specified by the PHB Unit ID and *config_addr*
Out	*Status*	0: Success -1: Hardware Error -3: Parameter or device enablement error

R1–7.3.4.2–2. The *ibm,write-pci-config* call must store the value to the configuration register which is at the location specified by the PHB Unit ID and *config_addr* in PCI configuration space.

R1–7.3.4.2–3. The *ibm,write-pci-config* call must perform a 1-byte, 2-byte, or 4-byte configuration space write depending on the value of the *size* input argument.

R1–7.3.4.2–4. The *config_addr* must be aligned to a 2-byte boundary if *size* is 2 and to a 4-byte boundary if *size* is 4.

R1–7.3.4.2–5. The *ibm,write-pci-config* call of IOAs or functions which are not present or which are not available to the caller must be ignored and a *Status* of 0 (Success) must be returned.

R1–7.3.4.2–6. For the LPAR option: The *Status* of -3 (Parameter or device enablement error) must be returned if all the following are true:

a. The OS attempts an *ibm,write-pci-config* to enable Memory or I/O for an IOA, without first calling *ibm,set-eeh-option* to enable EEH for the IOA

b. Enabling the IOA could expose other partitions to errors from the partition which is enabling the IOA

c. The hypervisor is enforcing EEH mode

Platform Implementation Note: In Requirement R1–7.3.4.2–6c, cross-partition errors could be caused due to error domains which are shared between the partitions. However, it is acceptable to share error domains when the IOA

and its device driver and the partition's OS cannot (through error or maliciously) cause errors which affect another partition.

7.3.5 Operator Interfaces and Platform Control

The RTAS operator interface and platform control functions provide the OS with the ability to perform platform services in a portable manner. The RTAS operator interface provides the ability for the OS to notify the user about OS events during boot, to notify the user of abnormal events, and to obtain information from the platform. The platform control functions give the OS the ability to obtain platform-specific information and to control platform features.

These calls are all “best effort” calls. RTAS should make its best effort to implement the intent of the call. If the Platform Hardware does not implement some optional feature, it is permitted for RTAS to either return an error, or to virtualize the service in some way and return “Operation Succeeded.”

Software Implementation Notes:

1. For example, a keyswitch could be virtualized by storing a keyswitch value in NVRAM and by providing a user interface to modify this value. The RTAS call *get-sensor-state* on the *keyswitch* returns the value stored in NVRAM.

2. If these services are only called prior to the use of any of the underlying devices by the OS, for example, during boot time, or only after the OS has finished using the devices, for example, during a crash, then the OS can avoid mutual exclusion and sharing concerns. Otherwise, synchronization per Section 7.2.3, “RTAS Critical Regions,” on page 109, must be performed.

7.3.5.1 Op Panel Display

R1–7.3.5.1–1. Platform Implementation: All servers must implement an operator panel display mechanism by supporting the *display-character* RTAS call.

Implementation Note: The operator display mechanism in Requirement R1–7.3.5.1–1 may be a physical alphanumeric display with a special purpose LCD device marked “used by RTAS”, or it may be some other virtualized display which is accessible through some method not defined by this architecture.

R1–7.3.5.1–2. Platform Implementation: Servers which provide *display-character* must provide a line length of at least 16 characters.

Software Implementation Notes:

1. There are currently four uses for the op panel display. The first is for display of an error code, if needed, from the Built-In-Self-Test (BIST) or Power-On-Self-Test (POST). This display is machine dependent. (These tests are executed prior to loading the OS or the operation of OF. Any display requirements are handled within the hardware.) The second is for progress indication during initialization and boot. This display is four digits and is updated as boot proceeds. The third is for display after a failure running diagnostics. In this case, a service request number (SRN) is displayed along with a FRU location code list of possible devices needing service. These numbers and locations can be longer than four characters. The SRN may be over 12 characters and a FRU location code list is one or more items, typically three, of 2 to 32 characters. The fourth is a crash code from the OS which is 12 characters indicating cause and dump status.

2. The RTAS *set-indicator* call with token #6 specifies 4 hex digits. The *display-character* call requires a minimum display size of one line of 4 characters, but a larger display may be made known to the OS using the “ibm,” extension properties defined in Section 7.3.5.3, “display-character,” on page 140. When the message to be displayed is larger than the OS believes the display to be, the OS should perform appropriate truncation, scrolling, or otherwise meaningfully display the message using the platform’s display resource.

3. Some servers implement a display larger than the default. For these servers, the **"ibm,display-line-length"** property and the **"ibm,display-number-of-lines"** property are set appropriately.

4. If the OS assumes the default display, the 2X16 display still works. It appears to be working in the bottom line and scrolling through the top line as long as only CR and LF are issued for control. The OF device tree properties indicate what is supported.

7.3.5.2 Service Processor

A service processor is not a platform requirement. Larger servers tend to be implemented with service processors. When implemented, the service processor is not seen directly as a device by software. All of its services which are visible to the OS, are abstracted with RTAS. The service processor may support the operator panel, manage sensors and indicators, run diagnostics, monitor the platform environment and save error logs. There is clearly an interface between RTAS and the service processor, but that interface is not intended to be used by the OS.

The service processor, in those platforms which choose to use one, is key in the initialization of the platform and has interfaces with the OF code. It is also involved with VPD collection and NVRAM access during initialization. It can also provide a serial port for a remote service capability. The service processor is also a significantly slower processor than the primary PA processor. Therefore, in the implementation of RTAS functions which use the service processor, care should be taken to avoid interlocks with the service processor which could significantly impair performance.

7.3.5.2.1 Surveillance

Platforms which include a service processor have the needed mechanism for a surveillance function; that is, the OS and the service processor can monitor each other. For example, if the OS crashes or hangs, or if the service processor has failures, a failure notification could occur. Notification could also occur if the platform fails during the boot process, or if it cannot complete a boot successfully. The notification can be sent to a service center or to a customer administrator, as determined by the customer setup of configuration parameters. The firmware provides notification to the OS by reporting exceptions through *event-scan.* The service processor can provide dial-out notification if the OS stops, or if a boot process fails.

In the implementation of surveillance, the service processor monitors the OS by tracking the issuance of heartbeats generated by calls to the *event-scan* RTAS service. If a service processor time-out occurs prior to receiving another heartbeat, an action based on user defined call out policy occurs. This action could be to reboot, call service or power-down. The policy may be different depending on whether the time-out occurs during a boot process or during a period of normal OS operation. The default policy and time-out period, kept in NVRAM, can be changed from a service processor menu or from software. The platform can be configured such that surveillance is either enabled or disabled immediately after boot. After boot, temporary changes to the surveillance state can be made by issuing a *set-indicator* call to indicator 9000 (see Section 7.3.5.4, "set-indicator," on page 142).

The following system parameters define the default behavior of surveillance mode (see also, Table 93, "Defined Parameters," on page 207 for more information about these parameters and for their default values).

- The **sp-sen** system parameter defines whether the default state of surveillance by the service processor is enabled (=on) or disabled (=off).
- The **sp-sti** system parameter defines the period of time (1-255 minutes) that the service processor should wait between heartbeats from *event-scan*. If the time-out period expires without the service processor receiving another heartbeat, the service processor initiates recovery and reporting actions as defined by the user.
- The **sp-sdel** system parameter defines the period of time (1-120 minutes) that the service processor should wait before starting surveillance after control passes to the OS. This value is set to allow enough time for the OS to boot and initialize to the point where it can start calling *event-scan* on a regular periodic basis.

Architecture Note: Surveillance times out if the time of the parameter, **sp-sdel**, plus the time of the parameter, **sp-sti,** passes prior to receiving the first heartbeat. In effect, the first *event-scan* can be considered the signal for boot complete.

The platform may perform surveillance on the service processor using *event-scan* to trigger checking as well as for reporting any errors found.

Software Implementation Note: The surveillance here is for keeping an eye on the overall functioning of the OS. If a specific process gets hung and the OS is still functioning, it is the responsibility of the OS to detect and not the surveillance discussed here.

OF Implementation Note: The OS is expected to call the *event-scan* RTAS service (with the internal-errors mask bit on) at the rate defined by the property **"rtas-event-scan-rate"** in the OF device tree. If an **"rtas-event-scan-rate"** of zero (0) is placed in the OF device tree and surveillance is initialized as 'active', a surveillance time-out occurs after the time-out period since the heartbeats are triggered by the *event-scan* call. If there is reason to operate with the rate = 0, the default state of surveillance (**sp-sen** parameter in NVRAM) should be disabled, and the surveillance sensor and indicator should not be placed in the OF device tree.

R1–7.3.5.2.1–1. Platform Implementation: The default surveillance policy must be defined by the **sp-sen**, **sp-sti** and **sp-sdel** system parameters, as set by the service processor or by software.

R1–7.3.5.2.1–2. Platform Implementation: Heartbeats to the service processor must only be sent as the result of a call to the *event-scan* RTAS service with the *internal-errors* bit (bit 0) set to 1 in the call buffer *Event Mask* parameter.

R1–7.3.5.2.1–3. Platform Implementation: In platforms which implement surveillance, the *event-scan* RTAS service may be called more than once per minute, but the heartbeat to the service processor must be sent at the rate of at least once per minute.

R1–7.3.5.2.1–4. Platform Implementation: In platforms which implement surveillance, the *ibm,os-term* RTAS call must be implemented.

Software Note: Requirement R1–7.3.5.2.1–4 provides a mechanism for the OS to release control of the platform without being aware of the state of surveillance. With the definition of a default platform state for surveillance, the OS may not be aware of the function, yet surveillance may be used. Platforms may not have a dependency on the OS to turn off surveillance during normal shutdown (a shutdown not including immediate reboot).

7.3.5.2.2 Surveillance on SMP Systems

Each running processor in an SMP system should be covered by surveillance. The following requirements assure this coverage.

R1–7.3.5.2.2–1. Each processor which is running, that is, not stopped by the *stop-self* RTAS call or not stopped due to BIST testing at bring-up, must issue the *event-scan* RTAS call. The rate of issue is the **"rtas-event-scan-rate"** times per minute divided by the number of processors. This is the minimum rate.

R1–7.3.5.2.2–2. The system must allow for all processors to cycle through their *event-scan* calls. The timeout period for a surveillance event, which is **sp-sti**, must be greater than n time t, where n is the number of processors and t is the **"rtas-event-scan-rate"**.

R1–7.3.5.2.2–3. The surveillance event must be signaled if after the surveillance interval, **sp-sti,** one or more processors has not issued an *event-scan* call.

Implementation Note: Care is required in the assignment of the surveillance interval and the **"rtas-event-scan-rate"** such that a surveillance event is not signaled prematurely. The default values are not meant for a system with a large number of processors.

7.3.5.3 *display-character*

The *display-character* function allows the display of both alphabetic and numeric information. The display for this function requires at least one line of four (4) characters. Also specified are the control characters carriage-return (CR) (0x0D) and line-feed (LF) (0x0A).

The following OF properties are defined in Section B.6.3.1, "RTAS Node Properties," on page 690:

- **"ibm,display-line-length"**
- **"ibm,display-number-of-lines"**
- **"ibm,display-truncation-length"**
- **"ibm,form-feed"**

R1–7.3.5.3–1. If *display-character* is implemented on a platform, the property **"ibm,display-line-length"** in the **/rtas** node must be provided if greater than the required minimum default of 4 characters.

R1–7.3.5.3–2. If *display-character* is implemented on a platform, the property **"ibm,display-number-of-lines"** in the **/rtas** node must be provided if greater than the required minimum default of 1 line.

R1–7.3.5.3–3. If the **"ibm,display-number-of-lines"** is greater than one, the platform must support form-feed (FF) (0x0C).

R1–7.3.5.3–4. If form-feed is implemented, it must clear the display and position the display pointer to line 1 column 1.

R1–7.3.5.3–5. The platform must include the property **"ibm,form-feed"** in the **/rtas** node.

R1–7.3.5.3–6. For the *display-character* RTAS call, when the truncation length as specified in the **"ibm,display-truncation-length"** property, when it exists, is less than the length of the line being displayed on that particular line, then the firmware must truncate the requested line to be displayed to the length specified in the **"ibm,display-truncation-length"** property for that line.

R1–7.3.5.3–7. For the *display-character* RTAS call, when the truncation length as specified in the **"ibm,display-truncation-length"** property, when it exists, is greater than the length specified of the line as specified in **"ibm,display-line-length"** then the platform must provide a platform-dependent method of displaying the line to the user.

R1–7.3.5.3–8. For platforms that use converged location codes, the platform must provide scrolling for the *display-character* RTAS call, on the second line of the display, and must provide the **"ibm,display-truncation-length"** property and specify a truncation length of no less than 80 characters for that line.

Platform and Software Implementation Note: In implementing Requirements R1–7.3.5.3–6 and R1–7.3.5.3–7, it is permissible to have a separate buffer for any of the lines of the display and not display that line until a button is pressed.

The RTAS call *display-character* can be used by the OS to display informative messages during boot, or to display error messages when an error has occurred and the OS cannot depend on its display drivers. This call is intended to display the alpha-numeric characters on an LCD panel, graphics console, or attached tty. The precise implementation is platform vendor specific.

R1–7.3.5.3–9. RTAS must implement a *display-character* call using the argument call buffer defined by Table 37, “display-character Argument Call Buffer,” on page 141 to place a character on the output device.

R1–7.3.5.3–10. The OS must serialize all calls to *display-character* with any other use of the *rtas-display-device*.

Table 37. *display-character* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *display-character*
	Number Inputs	1
	Number Outputs	1
	Value	Character to be displayed
Out	*Status*	0: Success -1: Hardware error -2: Device busy, try again later

R1–7.3.5.3–11. If a physical output device is used for the output of the RTAS *display-character* call, then it must have at least one line and 4 characters.

R1–7.3.5.3–12. Certain ASCII control characters must have their normal meanings with respect to position on output devices which are capable of cursor positioning. In particular, ^M (0x0D) must position the cursor at column 0 in the current line, and ^J (0x0A) must move the cursor to the next line. If on the bottom line, move to column 0 and scroll old data off the top.

R1–7.3.5.3–13. The ASCII characters which must be displayed are generally those coded from 0x20 to 0x7E as shown in Table 38, “Display ASCII Characters,” on page 141. SP indicates a space and ND is not defined

Table 38. Display ASCII Characters

Hex	Disp	Hex	Disp	Hex	Disp	Hex	Disp	Hex	Disp	Hex	Disp
20	SP	30	0	40	@	50	P	60	‘	70	p
21	!	31	1	41	A	51	Q	61	a	71	q
22	“	32	2	42	B	52	R	62	b	72	r
23	#	33	3	43	C	53	S	63	c	73	s
24	$	34	4	44	D	54	T	64	d	74	t
25	%	35	5	45	E	55	U	65	e	75	u
26	&	36	6	46	F	56	V	66	f	76	v
27	‘	37	7	47	G	57	W	67	g	77	w
28	(	38	8	48	H	58	X	68	h	78	x
29	)	39	9	49	I	59	Y	69	i	79	y
2A	*	3A	:	4A	J	5A	Z	6A	j	7A	z
2B	+	3B	;	4B	K	5B	[	6B	k	7B	{
2C	,	3C	<	4C	L	5C	\	6C	l	7C	|

Table 38. Display ASCII Characters *(Continued)*

Hex	Disp	Hex	Disp	Hex	Disp	Hex	Disp	Hex	Disp	Hex	Disp
2D	-	3D	=	4D	M	5D	]	6D	m	7D	}
2E	.	3E	>	4E	N	5E	^	6E	n	7E	~
2F	/	3F	?	4F	O	5F	_	6F	o	7F	ND

Software Implementation Note: Care should be taken in using the full character set for all systems as some characters may not be available or may display in a different fashion. For instance, the currency symbol, $ (0x24), may be modified to a national currency symbol. Other currently known differences occur for the reverse slant, \ (0x5C), and the tilde, ~(0x7E).

R1–7.3.5.3–14. RTAS must not output characters to the *rtas-display-device* except for explicit calls from the OS to the *display-character* function except for the following conditions.

a. The *rtas-display-device* is marked **`"used-by-rtas"`**.

b. The RTAS call is *power-off, ibm,power-off-ups, set-power-level (0,0),* or *system-reboot*.

Software Implementation Notes:

1. RTAS should try to produce output to the user. This could be to the system console, to an attached terminal, or to some other device. It could be implemented using a diagnostic processor or network. RTAS could also implement this call by storing the messages in a buffer in NVRAM so the user could determine the reason for a crash upon reboot.
2. This call modifies the registers associated with the *rtas-display-device*. The OS may also access this device, being aware that calls to *display-character* change the state of the device.

7.3.5.4 *set-indicator*

The RTAS *set-indicator* function provides the OS with an abstraction for controlling various lights, indicators, and other resources on a platform. If multiple indicators of a given type are provided by the platform, this function permits addressing them individually.

R1–7.3.5.4–1. RTAS must implement a *set-indicator* call which sets the value of the indicator of type *Indicator* and index *Indicator-index* using the argument call buffer defined by Table 39, “set-indicator Argument Call Buffer,” on page 142 and indicator types defined by Table 40, “Defined Indicators,” on page 143.

Table 39. *set-indicator* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *set-indicator*
	Number Inputs	3
	Number Outputs	1
	Indicator	Token defining the type of indicator
	Indicator-index	Index of specific indicator (0, 1,...)
	State	Desired new state

Table 39. *set-indicator* Argument Call Buffer

Parameter Type	Name	Values
Out	*Status*	990x: Extended Delay 0: Success -1: Hardware Error -2: Hardware busy, try again later -3: No such indicator implemented -9000: Multi-level isolation error -9001: Valid outstanding translation

R1–7.3.5.4–2. For indicators in the **"rtas-indicators"** property, the indices for indicators must start at zero (0) and increment sequentially up to the maximum index; that is, all of the integers and only those integers from 0 to the maximum index are valid.

Architecture Note: Indicator indices that are obtained via the *ibm,get-indices* RTAS call are not necessarily contiguous (that is, any of the indices between 0 and the *maxindex*, inclusive, may be missing).

R1–7.3.5.4–3. Of the indicator types defined by Table 40, "Defined Indicators," on page 143, RTAS must implement at least Tone Frequency and Tone Volume.

R1–7.3.5.4–4. The *set-indicator* RTAS call must not return a busy indication (-2 or 990x) for any indicator in Table 40, "Defined Indicators," on page 143 which is marked with a "yes" in the "Fast?" column of that table.

R1–7.3.5.4–5. The platform may, but is not required to, turn off a tone automatically after 5 minutes or more duration (that is, automatically set the Tone Volume to zero), and therefore a user of the Tone must call *set-indicator* Tone Volume with a volume value of non-zero, if a tone is to be sustained longer than 5 minutes, and if the platform is going to automatically terminate the tone, the platform must reset its automatic turn-off timer when it receives a *set-indicator* call for the Tone Volume with a non-zero tone volume value.

Table 40. Defined Indicators

Indicator Name	Token Value	Defined Values	Default Value	Fast?	Required?	<vendor>[a]	Examples/Comments
Tone Frequency	1	Unsigned Integer (units are Hz)	1000	yes	When tone is required. See Requirement R1–2.9–6	ibm	Generate an audible tone using the tone generator hardware. RTAS selects the closest implemented audible frequency to the requested value.
Tone Volume	2	0-100 (units are percent), 0 = OFF	0	yes	When tone is required. See Requirement R1–2.9–6	ibm	Set the percentage of full volume of the tone generator output, scaled approximately logarithmically. RTAS should select the closest implemented volume for values between zero (off) and 100 (full on).
-	3-6	-	-	-	-	-	Reserved.
-	7	-	-	-	-	-	Reserved. Was (deprecated) Battery Warning Time.
-	8	-	-	-	-	-	Reserved. Was (deprecated) Condition Cycle Request.
Surveillance	9000	0-disabled 1-255-timeout	sp-sti	yes	When the platform implements the surveillance function.	ibm	Initialized with value from the sp-sti system parameter.

Table 40. Defined Indicators ***(Continued)***

Indicator Name	Token Value	Defined Values	Default Value	Fast?	Required?	<vendor>[a]	Examples/Comments
Isolation-state	9001	Isolate = 0 Unisolate = 1	1	no	For all DR options	-	Isolate refers to the DR action to logically disconnect from the platform and/or OS (for example, for PCI, isolate from the bus and from the OS). See Section 13.5.3.4, “set-indicator,” on page 367 for more details.
DR	9002	Inactive = 0 Active = 1 Identify = 2 Action = 3	0 if Inactive 1 if Active	no	For all DR options	-	Indicator index may refer to a single indicator that combines Power/Active indicator and Identify/Action indications or just an Identify/Action indicator. Identify and Action may map to the same visual state (for example, the same blink rate). See Chapter 16, “Service Indicators,” on page 511 and Table 171, “Visual Indicator Usage,” on page 371 for more information.
Allocation-state	9003	unusable (0) usable (1) exchange (2) recover (3)		no	For all DR options	-	Allows an OS image to assign (usable, exchange, or recover) resources from the firmware or, release resources from the OS to the firmware. See Section 13.5.3.4, “set-indicator,” on page 367 for more details.
-	9004	-	-	-	-	-	Reserved.
Global Interrupt Queue Control	9005	Disable = 0 Enable = 1	1	yes	See Requirement R1–7.3.5.4.1.2–1	ibm	Enable and Disable the processor as Global Interrupt Queue Server
Error Log or FRU Fault	9006	Normal (off) = 0 Fault (on) = 1	0	no	Yes See Chapter 16, “Service Indicators,” on page 511	ibm	This indicator is combined with the Identify indicator for the Primary Enclosure drawer/enclosure (that is, is the same physical indicator). Off indicates that the system is working normally. On indicates that the system hardware, firmware and/or diagnostics detected a fault (failure) in the system or a partition requires operator intervention for another reason. The Error Log indicator is located only on the Primary Enclosure. See Chapter 16, “Service Indicators,” on page 511 and Table 171, “Visual Indicator Usage,” on page 371 for more information.
Identify (Locate)	9007	Normal (off) = 0 Identify (blink) = 1	0	no	Yes See Chapter 16, “Service Indicators,” on page 511	ibm	Note that a 9002 indicator also has an Identify state, and in the case where the 9002 indicator is implemented with two physical indicators (one for Power and one for Identify/Action), the same physical indicator must be used for both a 9002 Identify/Action indicator and 9007 Identify indicator. This architecture does not specify any mechanism for protecting against the simultaneous use by the user of an indicator that is both a 9002 and 9007 indicator, nor does it protect against the use of multiple 9007 indicators simultaneously or multiple uses of the same 9007 indicator simultaneously. See Chapter 16, “Service Indicators,” on page 511 and Table 171, “Visual Indicator Usage,” on page 371 for more information.
-	9008	-	-	-	-	-	Reserved.
-	9009	-	-	-	-	-	Reserved.
Vendor Specific	9100-9999					<vendor>[b]	Indicator values reserved for platform vendor use.

a. Values in the “<vendor>” column are used to replace the “<*vendor*>” field of the **"**<*vendor*>**,indicator-**<*token*>**"** property, when that property is presented. See Requirement R1–7.2.6.1.1–1.

b. The vendor specific company representation, as used on other OF properties specified by that vendor.

7.3.5.4.1 Indicators

7.3.5.4.1.1 Indicator 9000 Surveillance

An indicator is defined with the token value 9000 to allow temporary modification of the state of the surveillance function (further described in Section 7.3.5.2.1, “Surveillance,” on page 138).

To enable monitoring of heartbeats from the *event-scan* RTAS call, the surveillance indicator is set with a value of 1 to 255, indicating the number of minutes for the surveillance time-out value. If monitoring is already enabled, the time-out value can be modified by setting this indicator. To disable monitoring, the surveillance indicator should be set to a value of zero (0). The *set-indicator* call is used to modify the state of surveillance (overriding the default system parameter values) only for the current session. The surveillance state returns to the default values when the system is rebooted.

The default surveillance configuration may be modified by changing the system parameters. For more information on these parameters, refer to Section 7.3.5.2.1, “Surveillance,” on page 138.

R1–7.3.5.4.1.1–1. Platforms with the surveillance function must implement a sensor and an indicator, with the token value of 9000, with defined state input values of on (= 1-255, which enables surveillance with specified time-out value in minutes) and off (= 0, which disables surveillance).

Firmware Implementation Note: The requirement above results in the creation of the properties **`“ibm,indicator-9000”`** and **`“ibm,sensor-9000”`** in the **`/rtas`** node.

Hardware Implementation Note: The action that the service processor takes in the case of a timeout is determined by the configuration setup policy in the system parameters.

7.3.5.4.1.2 Indicator 9005 Global Interrupt Queue Control

The 9005 indicator controls the global interrupt server queue logic of the interrupt presentation controllers for the processor making the call (Available Processor Mask (APM) for the PowerPC interrupt presentation controller). This is used when bringing a processor online and taking a processor offline.

R1–7.3.5.4.1.2–1. Platforms that allow processors to be brought online or be taken offline dynamically must implement the global interrupt queue control indicator with a value of 9005 as specified in Table 40, “Defined Indicators,” on page 143.

R1–7.3.5.4.1.2–2. The index value for global interrupt queue control indicator (9005) must be ($2^{\texttt{ibm,interruptserver\#-size}}$) - 1 - the **`gserver#`** of the global server to be controlled as given in the **`“ibm,ppc-interrupt-gserver#s”`** property.

7.3.5.5 *get-sensor-state*

The RTAS call *get-sensor-state* is used by the OS to read the current state of various sensors on any Platform. If multiple sensors of a given type are provided by the platform, this function permits addressing them individually.

R1–7.3.5.5–1. RTAS must implement a *get-sensor-state* call which reads the value of the sensor of type *Sensor* which has index *Sensor-index* using the argument call buffer defined by Table 41, “get-sensor-state Argument Call Buffer,” on page 146 and the sensor types defined by Table 42, “Defined Sensors,” on page 147.

R1–7.3.5.5–2. If a platform tests sensor values against limits, then RTAS must return the result of these tests using the *Status* output parameter.

Table 41. *get-sensor-state* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for get-sensor-state
	Number Inputs	2
	Number Outputs	2
	Sensor	Token defining the sensor type
	Sensor-index	Index of specific sensor (0, 1,...)
Out	*Status*	990x: Extended Delay where x is a number 0-5 (see text below) 13: Sensor value >= Critical high 12: Sensor value >= Warning high 11: Sensor value normal 10: Sensor value <= Warning low 9: Sensor value <= Critical low 0: Success -1: Hardware Error -2: Hardware Busy, Try again later -3: No such sensor implemented -9000: DR Entity isolated (Chapter 13, “Dynamic Reconfiguration (DR) Architecture,” on page 355)
	State	Current value as defined in the Defined Values column of Table 42, “Defined Sensors,” on page 147

Software Implementation Note: When the 990x *Status* is returned, it is suggested that software delay for 10 raised to the x milliseconds (where x is the last digit of the 990x return code), before calling *get-sensor-state* again. However, software may issue the *get-sensor-state* call again either earlier or later than this.

R1–7.3.5.5–3. For sensors in the **`“rtas-sensors”`** property, the indices for sensors must start at zero (0) and increment sequentially up to the maximum index; that is, all of the integers and only those integers from 0 to the maximum index are valid.

Architecture Note: Sensor indices that are obtained via the *ibm,get-indices* RTAS call are not necessarily contiguous (that is, any of the indices between 0 and the *maxindex*, inclusive, may be missing).

R1–7.3.5.5–4. The *get-sensor* RTAS call must not return a busy indication (-2 or 990x) for any indicator in Table 42, “Defined Sensors,” on page 147 which is marked with a “yes” in the “Fast?” column of that table.

Hardware Implementation Note: Some platforms may compare the value of environmental sensors (such as the Battery Voltage or Thermal Sensor) to some limits. When the value of the sensor meets or exceeds a limit, the platform may take some action. RTAS makes the OS aware of the relationship of the sensor values to the limit by using the *Status* code to return this information.

Software and Hardware Implementation Notes: The meaning of these limits is as follows:

- Critical High - The sensor value is greater than or equal to this limit. The platform may take some action and may initiate an EPOW (see Section 10.2.2, “Environmental and Power Warnings,” on page 287). The OS may take some action to correct this situation or to perform an orderly shutdown.
- Warning High - The sensor value is greater than or equal to this limit, but less than the critical high limit. The platform may initiate a warning EPOW. The OS may take some action to bring this reading back into the normal range.
- Normal - RTAS is aware of the limits and the value is within these operating limits.

- Warning Low - The sensor value is less than or equal to this limit, but greater than the critical low limit. The platform may initiate a warning EPOW. The OS may take some action to bring this reading back into the normal range.
- Critical Low - The sensor value is less than or equal to this limit. The platform may take some action and may initiate an EPOW. The OS may take some action to correct this situation or to perform an orderly shutdown.

Where:

- A 'critical' state is defined as a condition where the sensor value of the measured item indicates that it is outside the allowable operating parameters of the system, and that a failure is imminent unless some immediate action is taken.
- A 'warning' state is defined as a condition where the sensor value of the measured item indicates that it is outside the expected operating parameters for normal operation, but has not yet reached a critical state. The variance is significant enough that either system software or an operator may want to take some action to bring the parameter back into the normal range.

Platform Implementation Note: The existence of this sensor state reporting capability should not be construed as a requirement to have any limits on sensors or to always have all four limits.

Table 42. Defined Sensors

Sensor Name	Token Value	Defined Values	Fast?	Required?	<vendor>[a]	Description
Key Switch	1	Off (0), Normal (1), Secure (2), Maintenance (3)	yes	No	ibm	Key switch modes are tied to OS security policy. Suggested meanings: Maintenance mode permits booting from floppy or other external, non-secure media. Normal mode permits boot from any attached device. Secure mode permits no manual choice of boot device, and may restrict available functionality which is accessed from the main operator station. Off completely disables the system.
-	2	-	-	-	-	Reserved.
Thermal	3	Temperature (in Degrees Celsius)	no	No	ibm	If implemented, returns the internal temperature of the specified thermal sensor.
-	4	-	-	-	-	Reserved. Was (deprecated) Lid Status.
-	5	-	-	-	-	Reserved.
-	6	-	-	-	-	Reserved. Was (deprecated) Current battery output voltage.
-	7	-	-	-	-	Reserved. Was (deprecated) Battery Capacity Remaining.
-	8	-	-	-	-	Reserved. Was (deprecated) Battery Capacity Percentage.
Environmental and Power State (EPOW)	9	EPOW_Reset(0) Warn_Cooling(1) Warn_Power(2) System_Shutdown(3) System_Halt(4) EPOW_Main_Enclosure(5) EPOW_Power_Off(7)	yes	Yes	ibm	RTAS assessment of the environment and power state of the platform.

Table 42. Defined Sensors ***(Continued)***

Sensor Name	Token Value	Defined Values	Fast?	Required?	<vendor>[a]	Description
-	10	-	-	-	-	Reserved. Was (deprecated) Battery Condition Cycle State.
-	11	-	-	-	-	Reserved. Was (deprecated) Battery Charging State.
Surveillance	9000	1-255 and 0	yes	When the platform implements the surveillance function	ibm	Current state of surveillance.
Fan speed	9001	fan - rpm	no	No	ibm	
Voltage	9002	voltage - mv	no	No	ibm	
DR-entity-sense	9003	DR connector empty = 0 DR entity present = 1 DR entity unusable (2) DR entity available for exchange (3) DR entity available for recovery (4)	no	For all DR options	-	Used in Dynamic Reconfiguration operations to determine if connector is available and whether the user performed a particular DR operation correctly. See Chapter 13, “Dynamic Reconfiguration (DR) Architecture,” on page 355 and Section 13.5.3.3, “get-sensor-state,” on page 366.
Power Supply	9004		no	No	ibm	Sense presence and status of power supplies.
Global Interrupt Queue Control	9005	Disabled = 0 Enabled = 1	yes	See Requirement R1–7.3.5.4.1.2–1	ibm	Global interrupt queue server control state.
Error Log or FRU Fault	9006	Normal (off) = 0 Fault (on) = 1	no	Yes See Chapter 16, “Service Indicators,” on page 511	ibm	Off indicates that the system is working normally. On indicates that the system hardware, firmware and/or diagnostics detected a fault in the system.
Identify	9007	Normal (off) = 0 Identify (blink) = 1	no	Yes See Chapter 16, “Service Indicators,” on page 511	ibm	Identify (locate) indicator (FRU, connector, or drawer/unit). Off is the default State. On indicates the Identify State.
-	9008	-	-	-	ibm	Reserved.
-	9009	-	-	-	ibm	Reserved.
Vendor Specific	9100-9999				<vendor>[b]	Reserved for use by platform vendors.

a. Values in the “<vendor>” column are used to replace the “<*vendor*>” field of the **“**<*vendor*>**,sensor-**<*token*>**”** property, when that property is presented. See Requirement R1–7.2.6.1.2–1.

b. The vendor specific company representation, as used on other OF properties specified by that vendor.

7.3.5.5.1 Sensors

The current state of surveillance, as described in Section 7.3.5.2.1, “Surveillance,” on page 138, is queried with a call to *get-sensor-state* with a token value of 9000. Fan speed is queried with the token value of 9001 and an index specifying the desired fan. Similarly, voltage is sensed with a token value of 9002 and an index specifying the desired voltage source.

R1–7.3.5.5.1–1. Platforms which implement the surveillance function must implement a single defined RTAS sensor with the token value of 9000, which returns values of on (= 1-255 minutes) and off (= 0) to show the current state of surveillance during this session.

R1–7.3.5.5.1–2. Platforms with software visible fan speed sensors must implement them as defined RTAS sensors with the token value of 9001, which returns a sensor value in revolutions per minute (RPM).

R1–7.3.5.5.1–3. Platforms with software visible voltage sensors must implement them as defined RTAS sensors with the token value of 9002, which returns a sensor value in millivolts.

Hardware Implementation Note: The notion of a delay, due to the sensor data acquisition time, may make it desirable to cache sensor data to avoid interlocking with the service processor.

Software Implementation Note: Software should not assume that sensor data returned is a real time reading.

7.3.5.5.1.1 Example Implementation of Sensors

An example implementation of a platform with a service processor and four fans and four voltage sensors is represented by the paired integers (*token maxindex*) in the OF device tree as shown in Table 43, “Example - Contents of “rtas-sensors” property,” on page 149.

Table 43. Example - Contents of **`“rtas-sensors”`** property

token	maxindex
(Any sensors with Standard Sensor Tokens) ...	(Associated max index values) ...
9000 (surveillance)	0000
9001 (fan-speed)	0003
9002 (voltage)	0003

This requires sensors such as those shown in Table 44, “Example - Sensor Definitions,” on page 149.

Table 44. Example - Sensor Definitions

sensor	token	index	value
surveillance	9000	0000	0 / 1-255
fan#1 fan speed	9001	0000	fan rpm
fan#2 fan speed	9001	0001	fan rpm
fan#3 fan speed	9001	0002	fan rpm
fan#4 fan speed	9001	0003	fan rpm
voltage-level #1	9002	0000	voltage - mv
voltage-level #2	9002	0001	voltage - mv
voltage-level #3	9002	0002	voltage - mv

Table 44. Example - Sensor Definitions *(Continued)*

sensor	token	index	value
voltage-level #4	9002	0003	voltage - mv

In addition, the properties **"ibm,sensor-9000"**, **"ibm,sensor-9001"** and **"ibm,sensor-9002"** in the **/rtas** node that each contain an array of strings. Each entry in the array contains the location code for the matching sensor. For example, the first entry of **"ibm,sensor-9001"** contains the location code for fan#1. Location codes are shown in Section 12.3, "Hardware Location Codes," on page 327. Of course, since it is an abstracted sensor, the entry for **"ibm,sensor-9000"** is NULL.

7.3.5.5.1.2 Power Supply Sensors

R1–7.3.5.5.1.2–1. Platforms with multiple software visible power supply sensors must implement them as defined RTAS sensors with the token value of 9004, which returns the values defined in Table 45, "Power Supply Sensor Values," on page 150.

Table 45. Power Supply Sensor Values

Value	Status
0	Not present
1	Present and Not operational
2	Status unknown
3	Present and operational

For static 9004 sensors, the maxindex in the **"rtas-sensors"** property for the token 9004 indicates the number of power supplies supported by the platform. In this case, the property **"ibm,sensor-9004"** in the **/rtas** node contains the location code for each index.

For dynamic 9004 sensors, the platform provides the information about the 9004 indicators as it would for other dynamic sensors. That is, the platform does not provide the **"ibm,sensor-9004"** property and instead provides the 9004 location code information through the *ibm,get-indices* RTAS call, and if the *ibm,get-indices* RTAS call returns an index of all-1's for a 9004 indicator, then the *ibm,get-dynamic-sensor-state* RTAS call is used to get the sensor state, instead of the *get-sensor* RTAS call.

7.3.5.5.1.3 Environmental Sensors

R1–7.3.5.5.1.3–1. Platforms which want to allow an application to analyze their environmental sensors must provide the property **"ibm,environmental-sensors"** in the **/rtas** node (see Section B.6.3.1, "RTAS Node Properties," on page 690).

The values for this property is a list of integers that are the token values (token) for the defined environmental sensors and the number of sensors (maxindex) for that token which are implemented on the platform.

Architecture Note: When a sensor is in the **"ibm,environmental-sensors"** property and when the sensor token indices are obtained via the *ibm,get-indices* RTAS call, the indices may not be contiguous for that sensor token (that is, any of the indices between 0 and the maxindex, inclusive, may be missing).

7.3.5.5.1.4 Sensor 9005 Global Interrupt Queue Control State

The 9005 sensor reports the state of the global interrupt server queue logic of the interrupt presentation controller for the specific processor making the call (Available Processor Mask (APM) for the PowerPC interrupt presentation controller). This is used when varying the processor on and off line.

R1–7.3.5.5.1.4–1. Platforms that allow processors to be brought online or be taken offline dynamically must implement the global interrupt queue control sensor with a value of 9005 as specified in Table 42, "Defined Sensors," on page 147.

R1–7.3.5.5.1.4–2. The index value for global interrupt queue control state sensor (9005) must be ($2^{\text{ibm,interrupt-server\#-size}}$) - 1- the gserver# of the global queue to be sensed as given in the **"ibm,ppc-interrupt-gserver#s"** property.

Note: on platforms that do not report **"ibm,interrupt-server#-size"** property, the assumed value of the size of the interrupt server number is 8.

7.3.6 Power Control

7.3.6.1 *set-power-level*

This RTAS call is used to set the power level of a power domain to either on or off.

R1–7.3.6.1–1. RTAS must implement the *set-power-level* call using the argument call buffer defined by Table 46, "set-power-level Argument Call Buffer," on page 151.

Table 46. *set-power-level* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *set-power-level*
	Number Inputs	2
	Number Outputs	2
	Power_domain	Token defining the power domain
	Level	Token for the desired level for this domain
Out	*Status*	0: Success -1: Hardware Error -2: Busy, Try again later 990x:Extended Delay where x is a number 0-5 (see text below)
	Actual_level	The power level actually set

R1–7.3.6.1–2. *Power_domain* must be a power domain identified in the OF device tree.

R1–7.3.6.1–3. *Level* must be 100 for full power and 0 for off.

R1–7.3.6.1–4. The *set-power-level* call must return the power level actually set in the *Actual_level* output parameter.

Software Implementation Notes:

1. The *set-power-level*(0,0) call, if implemented, removes power from the root domain, turning off power to all domains. The external events which can turn power back on are platform specific. The RTAS primitive *power-off* also removes power from the system, but permits specifying the events which can turn power back on.

2. The implemented values for the *Level* parameter for each power domain are defined in the OF device tree.

R1–7.3.6.1–5. The *set-power-level* RTAS call, when implemented, must return either a -2 or a 990x return code if the *set-power-level* operation specified in the RTAS call is going to exceed 1 millisecond in duration (where value of x gives a hint as to the duration of the busy; see text).

A single *set-power-level* operation may require an extended period of time for execution. Following the initiation of the hardware operation to change the power level, if the *set-power-level* call returns prior to successful completion of the operation, the call returns either a *Status* code of -2 or 990x. A *Status* code of -2 indicates that RTAS may be capable of doing useful processing immediately. A *Status* code of 990x indicates that the platform requires an extended period of time, and hints at how much time is required. Neither the 990x nor the -2 *Status* codes implies that the platform has initiated the operation, but it is expected that the 990x *Status* is used only if the operation had been initiated.

When the 990x *Status* is returned, it is suggested that software delay for 10 raised to the x milliseconds (where x is the last digit of the 990x return code), before calling *set-power-level* with the same power domain token. However, software may issue the *set-power-level* call again either earlier or later than this.

Software Implementation Note: In Requirement R1–7.3.6.1–5, a return code of -2 or 990x may either mean that the operation was initiated but not completed, or may mean that the operation was not initiated at all.

Firmware Implementation Notes:

1. If the RTAS initiates and returns before successful completion of the operation, then it needs to handle the split of a *set-power-level* operation across multiple calls.

2. It is the firmware's responsibility to not return a *Status* of 0 (success) until the operation is complete, and that may require performing an operation such as a delay operation or querying the hardware for power good status. In the former case, the firmware needs to save state between the calls to the same power domain number, until the operation is complete.

3. The *set-power-level* RTAS call may be called to set the power level of other power domains after the initiation to other domains and before the operation to those other domains are complete. If necessary, the *set-power-level* call may return a -2 or 990x *Status* to those calls without initiating the operation, if multiple simultaneous operations are not feasible.

7.3.6.2 *get-power-level*

R1–7.3.6.2–1. RTAS must implement the *get-power-level* call using the argument call buffer defined by Table 47, "get-power-level Argument Call Buffer," on page 153.

Table 47. *get-power-level* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *get-power-level*
	Number Inputs	1
	Number Outputs	2
	Power_domain	Token defining the power domain
Out	*Status*	0: Success -1: Hardware Error -2: Busy, try again later -3: Can't determine current level
	Level	The current power level for this domain

R1–7.3.6.2–2. *Power_domain* must be a power domain identified in the OF device tree.

Software Implementation Note: The *get-power-level* call only returns information about power levels whose state is readable in hardware. It does not need to remember the last set state and return that value.

7.3.6.3 *power-off*

This primitive turns power off on a system which is equipped to perform a software-controlled power off function.

R1–7.3.6.3–1. If software controlled power-off hardware is present, the *power-off* function must turn off power to the platform, using the argument call buffer described in Table 48, "power-off Argument Call Buffer," on page 153.

Table 48. *power-off* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *power-off*
	Number Inputs	2
	Number Outputs	1
	Power_on_mask_hi	Mask of events that can cause a power on event - event mask values [0:31] (right-justified if the cell size is 64 bits)
	Power_on_mask_lo	Mask of events that can cause a power on event - event mask values [32:63] (right-justified if the cell size is 64 bits)
Out	*Status*	On successful operation, does not return -1: Hardware error

R1–7.3.6.3–2. If software controlled power-off hardware is present, *Power_on_mask,* which is passed in two parts to permit a possible 64 events even on 32-bit implementations, must be a bit mask of power on triggers, or if the **`"power-on-triggers"`** property is absent from the **`/rtas`** node, a value of 0 must be used for *Power_on_mask_hi* and *Power_on_mask_lo*.

R1–7.3.6.3–3. Platforms must omit the **`"power-on-triggers"`** property from the **`/rtas`** node.

Implementation Note: The power on triggers, which were removed from this architecture, are documented in the Table 49, “Defined Power On Triggers,” on page 154, for legacy reasons.

Table 49. Defined Power On Triggers

Bit	Event
0	Power Switch On
2	Lid Open
5	Wake Button
8	Switch to Battery
9	Switch to AC
10	Keyboard or mouse activity
12	Enclosure Closed
13	Ring Indicate
14	LAN Attention
15	Time Alarm
16	Configuration change
17	Service Processor

R1–7.3.6.3–4. For the System Parameters option: If software controlled power-off hardware is present, the power-off function must prevent reboot in the event of a later external power recovery with the platform_auto_power_restart system parameter enabled.

7.3.6.4 ibm,power-off-ups

This RTAS call manages the system power-off function in systems which may have power backed up with an Uninterruptible Power Supply (UPS).

R1–7.3.6.4–1. For platforms that support a platform controlled Uninterruptible Power Supply (UPS), the *ibm,power-off-ups* function must be implemented, whether a platform controlled UPS is present or not, using the argument call buffer described in Table 50, “ibm,power-off-ups Argument Call Buffer,” on page 154.

Table 50. *ibm,power-off-ups* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,power-off-ups*
	Number Inputs	0
	Number Outputs	1
Out	*Status*	On successful operation, does not return -1: Hardware error

R1–7.3.6.4–2. If a platform controlled UPS is present, then the *ibm,power-off-ups* RTAS call must turn off system power while enabling platform auto restart upon restoration of system power, according to the platform_auto_power_restart policy described in Section 7.3.16.5.2, “platform_auto_power_restart Parameter,” on page 221, and must not return, otherwise, the call must not turn off system power and must not return.

R1–7.3.6.4–3. If a platform controlled UPS is not present, then the *ibm,power-off-ups* RTAS call must turn off system power while enabling platform auto restart upon restoration of system power, according to the platform_auto_power_restart policy described in Section 7.3.16.5.2, “platform_auto_power_restart Parameter,” on page 221, and must not return, otherwise, the call must not turn off system power and must not return.

Software Implementation Notes:

1. Supporting *ibm,power-off-ups*, allows a system to be shutdown due to a report that the system was running under UPS power for systems with a platform managed UPS. As opposed to *power-off*, *ibm,power-off-ups*, permits the operating system to be restarted when power is restored after a loss of external power.
2. The report that a system needs to be shutdown due to running under a UPS would be given by the platform as an EPOW event with EPOW event modifier being given as, 0x02 = Loss of utility power, system is running on UPS/Battery, as described in section Section 10.3.2.2.8, “Platform Event Log Format, EPOW Section,” on page 308.
3. If the RTAS *ibm,power-off-ups* call is supported by the platform, it will also allow a shutdown with a subsequent restart when power is restored for systems running with a UPS that is not under platform control. This presumes that the OS has some external means of recognizing when running under UPS power to initiate the *ibm,power-off-ups* call.

7.3.7 Reboot and Flash Update Calls

During execution, it may become necessary to shut down processing and reboot the system in a new mode. For example, a different OS level may need to be loaded, or the same OS may need to be rebooted with different settings of System Environment Variables.

7.3.7.1 *system-reboot*

R1–7.3.7.1–1. RTAS must implement a *system-reboot* call which resets all processors and all attached devices. After reset, the system must be booted with the current settings of the System Environment Variables (refer to Section 8.4.1, “System (0x70),” on page 267 for more information).

R1–7.3.7.1–2. The RTAS *system-reboot* call must be implemented using the argument call buffer defined by Table 51, “system-reboot Argument Call Buffer,” on page 155.

Table 51. *system-reboot* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *system-reboot*
	Number Inputs	0
	Number Outputs	1
Out	*Status*	On successful operation, does not return -1: Hardware error

Hardware Implementation Note: The platform must be able to perform a system reset and reboot. On a multiprocessor system, this should be a hard reset to the processors.

7.3.7.2 *ibm,update-flash-64-and-reboot*

The *ibm,update-flash-64-and-reboot* function is described in this section. It does not return to the OS if successful. This call supports RTAS instantiated in 32 bit mode to access storage at addresses above 4GB. In an exception to the LPAR Requirement R1–14.6–6 this call supports block lists being outside of the Real Mode Area (RMA) as long as the initial block list is at an address below the limits of the cell size of the *Block_list* argument.

R1–7.3.7.2–1. The argument call buffer for the *ibm,update-flash-64-and-reboot* RTAS call must correspond to the definition in Table 52, “ibm,update-flash-64-and-reboot Argument Call Buffer,” on page 156.

Table 52. *ibm,update-flash-64-and-reboot* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,update-flash-64-and-reboot*
	Number Inputs	1
	Number Outputs	1
	Block_list	A real pointer to a block list of 64 bit entries
Out	*Status*	-1: Hardware error -3: Image unacceptable to update program -4: Programming failed when partially complete, and the flash is now corrupted - reboot may fail -9002: Not authorized
The *Status* of 0 is never returned, because this RTAS call does not return if successful.		
The -1 return is to cover the case where some condition prevents RTAS from being able to program the flash at this time. For example, the flash programming power supply is disconnected, a low-level security check (for instance a switch or jumper) fails, or a test programming probe fails for an unknown reason or the case where the flash has been successfully updated, but the reboot fails for some reason.		
The -3 return is to cover the case where embedded vendor/platform specific information in the image failed to conform to the required format or content for this platform, such as the firmware revision number or a CRC or some other check which was intended to ensure the integrity of the image.		
The -4 return is to cover the case where the update failed before the image was fully updated. In this case, the OS has the responsibility for reporting the failure.		
The -9002 return code is used to indicate that the partition at the time the call was made was not authorized to update the flash image.		

R1–7.3.7.2–2. The RTAS *ibm,update-flash-64-and-reboot* call *Block_list* on platforms that do not present the **`“ibm,flash-block-version”`** property in the OF **`/rtas`** node must conform to the definition shown in Table 53, “Format of Block List,” on page 156.

Table 53. Format of Block List

Length of *Block_list* in bytes
Address of memory area 1
Length of memory area 1
. . .

Table 53. Format of Block List

Address of memory area n
Length of memory area n

R1–7.3.7.2–3. The *ibm,update-flash-64-and-reboot* RTAS call *Block_list* must be a sequence of 64 bit cells.

R1–7.3.7.2–4. Memory blocks referenced in the *ibm,update-flash-64-and-reboot* RTAS call *Block_list* must reside in System Memory outside that reserved for firmware (both the RTAS data area and OF's memory defined by real-base and real-size).

R1–7.3.7.2–5. The block list referenced by the *Block_list* argument to the *ibm,update-flash-64-and-reboot* RTAS call must be in System Memory below the maximum address supported by the RTAS instantiated cell size.

R1–7.3.7.2–6. The addresses of memory blocks referenced by the *ibm,update-flash-64-and-reboot* RTAS call *Block_list* must align tn a 4 KB boundary.

R1–7.3.7.2–7. A memory block, included in the *ibm,update-flash-64-and-reboot* RTAS call *Block_list,* must not cross a 256MB boundary.

R1–7.3.7.2–8. The *ibm,update-flash-64-and-reboot* call must test the image to make sure it has the right format and is not damaged, update the flash from the *Block_list* and then perform a system reset and reboot, as for the *system-reboot* call.

Hardware and Software Implementation Note: Platform specific information should be embedded in the flash images to identify the firmware unambiguously and to ensure that the firmware operates correctly on the platform. Such information might include platform board model and revision numbers covered by the firmware, manufacturer ID, and firmware revision number used for external display. This information should include a CRC or other check which ensures the integrity of the data.

Software Implementation Notes:

1. The execution time for this calls may be in the order of seconds, rather than "a few tens of microseconds" as noted on page 111.
2. The RTAS flash update programs should display progress, completion, and error information while the flash update is underway, if possible.
3. The OS does not expect a return from the *ibm,update-flash-64-and-reboot* call other than for cases where the hardware cannot be accessed, the flash image is unacceptable to the RTAS flash update program, the result of the update corrupted the flash, or the platform could not be rebooted.

7.3.7.3 Flash Update with Discontiguous Block Lists

The property **`"ibm,flash-block-version"`** (see Section B.6.3.1, "RTAS Node Properties," on page 690) is defined to describe the following definition and operation of the *Block_list* shown in Table 54, "Format of Discontiguous Block_list," on page 157.

Table 54. Format of Discontiguous *Block_list*

VER	Length of *Block_list* in bytes
Address of *Block_list* extension	
Address of memory area 1	

Table 54. Format of Discontiguous *Block_list*

Length of memory area 1
Address of memory area 2
Length of memory area 2
- - -
Address of memory area n
Length of memory area n

Where:

- VER (1 byte in length) indicates the version of the *Block_list*.
- Length of the *Block_list* in bytes indicates the size of this *Block_list*, including the header cell and the cell with the address of the *Block_list* extension.
- Address of the *Block_list* extension indicates the location of the next *Block_list*. 0x00 indicates no additional *Block_list* extension.
- Address of memory area 1 (2 . . . n) indicates the location of this portion of the flash image.
- Length of memory area 1 (2 . . . n) indicates the length of this portion of the flash image.

R1–7.3.7.3–1. If VER is 0x01, the *Block_list* must be formatted as in Table 54, “Format of Discontiguous Block_list,” on page 157.

R1–7.3.7.3–2. If VER is 0x01, the *Block_list* parameter in the function call or the Address of the *Block_list* extension, if not 0x00, must point to a *Block_list* cell containing VER and Length of the *Block_list*.

R1–7.3.7.3–3. If VER is 0x01, the Address of the *Block_list* extension parameter must be 0x00 to indicate that there are no further extensions.

R1–7.3.7.3–4. The VER byte must exist in the *Block_list* and in each *Block_list* extension.

R1–7.3.7.3–5. If the platform supports the property **`“ibm,flash-block-version”`** with value 0x01, it must also support the default value 0x00.

The *Block_list* format allows flexibility in the size and page requirements for the block lists. Page alignment is not required for the lists or extensions. They may run across contiguous pages with the control being the length of each list or extension and with the end being the 0x00 pointer.

7.3.7.4 ibm,manage-flash-image

The *ibm,manage-flash-image* RTAS call supports systems having a “temporary” and “permanent” flash image areas. It allows the user to commit the temporary flash image by copying it to the permanent image area. It also allows the user to reject the temporary flash image by overwriting it with the permanent flash image.

R1–7.3.7.4–1. The RTAS *ibm,manage-flash-image* call must be implemented using the argument call buffer defined by Table 55, “ibm,manage-flash-image Argument Call Buffer,” on page 159.

Table 55. *ibm,manage-flash-image* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for ibm,manage-flash-image
	Number of Inputs	1
	Number of Outputs	1
	Image to Commit	0: Reject “temporary” firmware image 1: Commit “temporary” firmware image
Out	*Status*	0: Success -1: Hardware Error -2: Busy -3: Parameter Error -9001: Cannot Overwrite the Active Firmware Image Error -9002 Not Authorized 990x: Extended Delay

R1–7.3.7.4–2. The *ibm,manage-flash-image* RTAS call must not change the system flash and must return a *Status* of value -9001 when called with a request to reject the temporary firmware image when not running on the permanent firmware image.

R1–7.3.7.4–3. The *ibm,manage-flash-image* RTAS call must not change the system flash and must return a *Status* of value -9001 when called with a request to commit the temporary firmware image when not running on the temporary firmware image.

Platform Implementation Note: In platforms supporting two firmware image areas, platforms always apply updates to the temporary image area. The RTAS call *ibm,manage-flash-image* is the normal means by which a temporary image is committed to the permanent side. However, if a platform is running from a temporary image when an update is to be applied, then the platform may automatically commit the current temporary image to the permanent side to allow the new image to be updated to the temporary image area. The *ibm,validate-flash-image* RTAS call is used to determine what would result from an attempt to update a FLASH image taking in to account the image to be updated and the current image being executed.

7.3.7.5 ibm,validate-flash-image

The *ibm,validate-flash-image* RTAS call allows OS service code to determine if a candidate flash image is valid, if the partition has authority to update the flash image, and what the resulting flash levels will be after performing the update.

R1–7.3.7.5–1. The *ibm,validate-flash-image RTAS* call must be implemented using the argument call buffer described in Table 56, “ibm,validate-flash-image Argument Call Buffer,” on page 160.

Table 56. *ibm,validate-flash-image* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,validate-flash-image*
	Number of Inputs	2
	Number of Outputs	2
	Buffer Ptr	Real address of minimum 4 K buffer, contiguous in real memory
	Buffer Size	Size in bytes of Buffer
Out	*Status*	990x: Extended Delay 0: Success -1: Hardware Error -2: Busy -3: Parameter Error -9002: Not authorized
	Update Results Token	Token to identify what will happen if update is attempted with this token, described in Requirement R1–7.3.7.5–6.

R1–7.3.7.5–2. The *ibm,validate-flash-image RTAS* call *Buffer Ptr* parameter must be a real address representing the starting address of a minimum 4 K buffer, contiguous in real memory.

R1–7.3.7.5–3. On input, the *ibm,validate-flash-image RTAS* call buffer pointed to by the *Buffer Ptr* parameter must contain the first 4 KB of the candidate flash image to be validated.

R1–7.3.7.5–4. For the LPAR option: The *ibm,validate-flash-image RTAS* call buffer described in Requirement R1–7.3.7.5–2 must be in the partition's RMA.

R1–7.3.7.5–5. On exit from the *ibm,validate-flash-image RTAS* call, RTAS must place the following data in the buffer, starting at the address in the *Buffer Ptr* parameter:

- “MI” <sp> current-T-image <sp> current-P-image <0x0A>
- “MI” <sp> new-T-image <sp> new-P-image <0x00>
- “ML” <sp> current-T-image <sp> current-P-image <0x0A>
- “ML” <sp> new-T-image <sp> new-P-image <0x00>

In Requirement R1–7.3.7.5–5, current-T-image and current-P-image are the fixpack microcode image names currently on the Temporary and Permanent sides, respectively, and new-T-image and new-P-image are the fixpack microcode image names that will exist in flash after a successful flash update with the candidate image.

If the current flash image level is not known, the value provided for current-T-image and/or current-P-image is “UNKNOWN”.

If the flash update function would not succeed, the values of new-T-image and new-P-image are the same as current-T-image and current-P-image, respectively.

R1–7.3.7.5–6. On exit from the *ibm,validate-flash-image RTAS* call, the *Update Results Token* output must be updated with one of the values in Table 57, “Update Results Token Values,” on page 161. This list is in order; firmware must provide the first value in the list which would be true if an update is attempted:

Table 57. Update Results Token Values

Token	Description
1	No update done, partition does not have authority to perform flash update
2	No update done, the candidate image is not valid for this platform
3	Current fixpack level is unknown, the new-T-image and new-P-image identifies show what will exist in flash after update with this image
4	Current T side will be committed to P side before being replace with new image, and the new image is downlevel from current image
5	Current T side will be committed to P side before being replaced with new image
6	T side will be updated with a downlevel image
0	T side will be updated with a newer or identical image

7.3.7.6 ibm,activate-firmware

The *ibm,activate-firmware* allows an OS to activate a new version of firmware that has been updated in the platform flash memory after the partition was started.

R1–7.3.7.6–1. The *ibm,activate-firmware* RTAS call must be implemented using the argument call buffer described in Table 58, “ibm,activate-firmware Argument Call Buffer,” on page 161.

Table 58. *ibm,activate-firmware* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,activate-firmware*
	Number of Inputs	0
	Number of Outputs	1
Out	*Status*	990x: Extended Delay 0: Success -1: Hardware Error -2: Busy, try again later -3: Parameter Error -9001: No valid FW available to activate

Software implementation Note: The OS should expect that a number of calls may be required to accomplish firmware activation, with “Busy, try again later” or “Extended Delay” return codes from all but the last call. The new version of firmware is not in use until a “Success” return. The OS may interleave calls to other RTAS functions between calls to this function.

7.3.8 SMP Support

In a Symmetric Multiprocessor (SMP) system, the platform needs the ability to synchronize the clocks on all the processors. The timebase registers are synchronized by the platform before CPUs are given to the OS.

R1–7.3.8–1. *(Merged into Requirement R1–11.1–9)*

R1–7.3.8–2. *(Merged into Requirement R1–11.1–9)*

7.3.8.1 *stop-self*

The stop-self primitive causes a processor thread to stop processing OS or user code, and to enter a state in which it is only responsive to the *start-cpu* RTAS primitive. This is referred to as the RTAS *stopped* state.

R1–7.3.8.1–1. A *stop-self* RTAS call must place the calling processor thread in the RTAS *stopped* state. This call must be implemented using the argument call buffer defined by Table 59, “stop-self Argument Call Buffer,” on page 162.

R1–7.3.8.1–2. RTAS must insure that a processor thread in the RTAS *stopped* state does not checkstop or otherwise fail if a machine check or soft reset exception occurs. Processor threads in this state receive the exception, but must perform a null action and remain in the RTAS *stopped* state.

Table 59. *stop-self* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *stop-self*
	Number Inputs	0
	Number Outputs	1
Out	*Status*	If successful, this call does not return -1: Hardware Error

Software Implementation Note: If this call succeeds, it does not return. The CPU thread waits for some other processor thread to issue a *start-cpu* targeted to this processor thread.

Firmware Implementation Note: In an LPAR environment the state of the interrupt sub-system associated with this processor on entry to this call cannot be trusted. Although interrupts are masked as part of the RTAS call protocol, the caller may have left the processor configured as an interrupt server. Therefore, interrupt signals may be pending within the processor’s interrupt management area. These conditions need to be cleared prior to allocating this processor to another partition.

R1–7.3.8.1–3. Platforms which support the enhanced *stop-self* RTAS behavior must include the name only **`"ibm,integrated-stop-self"`** OF property, under the **`/rtas`** node, and prior to placing a processor in the stopped state, flush and disable any caches/memory exclusively used by the issuing processor.

Architecture Note: In Requirement R1–7.3.8.1–3, an exclusively used cache means that no other running processor currently needs the cache for normal operation, even if the cache could potentially be shared with other processors. An exclusively used memory means any main memory allocated local to the processor thread and thus not accessible by other processor threads.

R1–7.3.8.1–4. Execution of the *stop-self* call by the last active processor thread must cause the firmware to recover all the resources owned by the executing OS image for use per platform policy.

7.3.8.2 *start-cpu*

The *start-cpu* primitive is used to cause a processor thread which is currently in the RTAS *stopped* state to start processing at an indicated location.

R1–7.3.8.2–1. A *start-cpu* RTAS call must remove the processor thread specified by the *CPU_id* parameter from the RTAS *stopped* state. This call must be implemented using the argument call buffer defined by Table 60, “start-cpu Argument Call Buffer,” on page 163.

R1–7.3.8.2–2. The processor thread specified by the *CPU_id* parameter must be in the RTAS *stopped* state entered because of a prior call by that processor to the *stop-self* primitive.

R1–7.3.8.2–3. When a processor thread exits the RTAS *stopped* state, it must begin execution in real mode, with the MSR in the same state as from a system reset interrupt (except for the MSR_{HV} bit which is on if not running under a hypervisor and off if running under a hypervisor) at the real location indicated by the *Start_location* parameter, with register R3 set to the value of parameter *Register_R3_contents* and the MSR as defined in Table 61, “Machine State Register (MSR) State in Started Processor,” on page 163. All other register contents are indeterminate.

Table 60. *start-cpu* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *start-cpu*
	Number Inputs	3
	Number Outputs	1
	Cpu_id	Token identifying the processor thread to be started, obtained from the value of the **“ibm,ppc-interrupt-server#s”** property for the CPU in the OF device tree
	Start_location	Real address at which the designated CPU begins execution
	Register_R3_contents	Value which is loaded into Register R3 before beginning execution at Start_location
Out	*Status*	0: Success -1: Hardware Error

Note: Requirement R1–5.2.6–2 applies to the *start-cpu* RTAS call. At the completion of *start-cpu*, the caches to be used by the specified processor must have been initialized and the state bits made accurate prior to beginning execution at the start address.

Table 61. Machine State Register (MSR) State in Started Processor

Bit Number	Name	Initial Value upon start by start-cpu	Bit Number	Name	Initial Value upon start by start-cpu
0	SF	0 if RTAS instantiated in 32 bit mode 1 if RTAS instantiated in 64 bit mode	53	SE	0
1	Reserved	0	54	BE	0
2	Reserved	0	55	FE1	Implementation Dependent
3	HV	0 if running under hypervisor firmware, 1 if running in “SMP” mode	56	US	0
4:46	Reserved	0	57	Reserved	0
47	ILE	0	58	IR	0

Table 61. Machine State Register (MSR) State in Started Processor

Bit Number	Name	Initial Value upon start by start-cpu	Bit Number	Name	Initial Value upon start by start-cpu
48	EE	0	59	DR	0
49	PR	0	60	Reserved	0
50	FP	Implementation Dependent	61	PMM	0
51	ME	1	62	RI	0
52	FE0	Implementation Dependent	63	LE	0

7.3.8.3 *query-cpu-stopped state*

The *query-cpu-stopped-state* primitive is used to query a different processor thread to determine its status with respect to the RTAS stopped state.

R1–7.3.8.3–1. A *query-cpu-stopped-state* RTAS call must return the *CPU_status* of the processor thread specified by the *Cpu_id* parameter. This call must be implemented using the argument call buffer defined by Table 62, “query-cpu-stopped-state Argument Call Buffer,” on page 164.

Table 62. *query-cpu-stopped-state* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *query-cpu-stopped-state*
	Number Inputs	1
	Number Outputs	2
	Cpu_id	Token identifying the processor thread to be queried, obtained from the value of the **“ibm,ppc-interrupt-server#s”** property for the CPU in the OF device tree
Out	*Status*	0: Success -1: Hardware Error -2: Hardware Busy, Try again later
	CPU_status	0: The processor thread is in the RTAS stopped state 1: *stop-self* is in progress 2: The processor thread is not in the RTAS stopped state

Firmware Implementation Note: RTAS serialization may be required between the *stop-self* and the *query-cpu-stopped-state* calls.

Software Implementation Note: The OS performs a *stop-self* on the desired processor thread, then periodically call*s* *query-cpu-stopped-state* on another processor thread until the desired processor thread is stopped. Before calling

set-power-level to power off the desired processor, or isolate the logical CPU, the platform requires that all processor threads be in the RTAS stopped state.

7.3.9 Miscellaneous RTAS Calls

7.3.9.1 *ibm,os-term*

This RTAS call is provided for the OS to indicate to the platform that it has terminated normal operation. A string of information is passed to the platform.

A call to the *ibm,os-term* RTAS function implies the following to the platform:

- Any platform reporting and recovery policies may now take effect.
- The OS may no longer be issuing periodic *event-scan* requests, so surveillance monitoring does not continue.
- All devices not marked **"used-by-rtas"** are released by the OS (including, for example, native serial ports used by a service processor).
- The OS no longer responds to any EPOW events, so it is up to the platform to take any appropriate actions for such events.

Due to the above implications, the platform may take actions (for example, a service processor "call home") that could conflict with normal processing of further RTAS requests. However, since the OS has entered a "live halt" state, the list of RTAS functions that it still needs is relatively small. The list of RTAS functions that the platform might expect to see after *ibm,os-term* includes:

- *nvram-fetch*
- *nvram-store*
- *display-character*
- *power-off*
- *ibm,power-off-ups*
- *system-reboot*
- *check-exception* for machine checks (Although the OS may still react normally to a machine check condition by calling *check-exception*, it might not process a returned error log. It is allowable for *check-exception* to not return an extended log when in this state.)

If a platform has a service processor, and a policy has been established for actions to be taken by the service processor upon receiving notice of OS termination, the service processor may complete those actions and a return to the CPU from this call may never occur. If the call does return, the OS performs its own termination policy.

When the platform supports extended *ibm,os-term* behavior, the return to the RTAS will always occur unless there is a kernel assisted dump active as initiated by an *ibm,configure-kernel-dump* call.

Platforms capable of supporting this extended *ibm,os-term* behavior will so indicate by presenting the **"ibm,extended-os-term"** RTAS property in the OF device tree.

R1–7.3.9.1–1. RTAS must implement an *ibm,os-term* call using the argument call buffer defined by Table 63, "ibm,os-term Argument Call Buffer," on page 166 to receive a termination string from the OS.

Table 63. *ibm,os-term* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,os-term*
	Number Inputs	1
	Number Outputs	1
	Pointer to String	NULL terminated string
Out	*Status*	0:success -1:hardware error -2:hardware busy, try again later

Platform Implementation Note: The string should be maintained in an error log which could be made accessible to a service location or saved in the platform for later remote access.

R1–7.3.9.1–2. The *ibm,os-term* call must disable surveillance.

R1–7.3.9.1–3. During the machine check and soft reset handlers, the platform must support access to the *ibm,os-term* RTAS function.

R1–7.3.9.1–4. If the *ibm,os-term* call does not return to the caller, the platform must honor the partition_auto_restart system parameter value.

R1–7.3.9.1–5. For platforms supporting extended *ibm,os-term* behavior, the *ibm,os-term* call must always return unless there is an active kernel assisted dump configured as specified by an *ibm,configure-kernel-dump* RTAS call.

Platform implementation note: The *ibm,os-term* RTAS call allows for the case where the OS and platform may share an I/O device such as a TTY where the OS would have use of the device normally, and the platform use when the OS has terminated, such as to implement an error reporting call home function both in the OS and the platform. For proper sharing in such a case where extended behavior is supported, when the primary partition console is also used for the call-home by the platform, the platform should not initiate the call home until after the partition shuts down.

7.3.9.2 *Ibm,exti2c*

For support of platforms which require an external I^2C bus, a special port to the service processor is required. The EXTI2C option is designed to control specific external devices. Designers cannot assume that an arbitrary I^2C device may be substituted.

The *ibm,exti2c* call provides a single channel to the I^2C bus. Through this channel, software can read or write up to 256 bytes from/to an addressed resource within an address space between X'000000 and X'FFFFFF. Reference the specification for the specific I^2C device to determine what effect such operations may have.

The Buffer Pointer argument is used to manage this channel across multiple *ibm,exti2c* RTAS calls. If the input Buffer Pointer value on a call is zero, the state of the channel is reset and any outstanding I^2C operation is aborted. If the input Buffer Pointer has a different value from that of the last call, a new operation is started, with any previous operation being aborted. An input Buffer Pointer value that is the same as that used on the previous call indicates a continuation of the last operation, given that the *Status* of the last call was not 0 (success) or -1 (hardware error). These terminating statuses reset the channel.

Using software must manage serialization to the *ibm,exti2c* channel across multiple calls for the same I^2C operation.

A single *ibm,exti2c* operation may require an extended period of processing by background hardware. During this time, RTAS returns either a *Status* code of -2 or 990x. A *Status* of -2 indicates that RTAS may be capable of doing useful processing immediately.

A *Status* code of 990x indicates that the platform requires an extended period of time to perform the operation. It is suggested that software delay for 10 raised to the x milliseconds before calling *ibm,exti2c* with the same Buffer Pointer value, however, software may call again earlier or later.

A *Status* code of -1 indicates either a general error associated with the local I^2C hardware (service processor) or that the channel has been corrupted due to other error conditions not associated with the I^2C operation. If the buffer is changed, as when an error code is returned, the RTAS *Status* code is 0 (success).

R1–7.3.9.2–1. For the EXTI2C option: RTAS must implement an *ibm,exti2c* call using the argument call buffer defined by Table 64, “ibm,exti2c Argument Call Buffer,” on page 167 to allow communications with special hardware.

Table 64. *ibm,exti2c* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,exti2c*
	Number Inputs	1
	Number Outputs	1
	Buffer Pointer	Real Address of data buffer
Out	*Status*	0:Success -1:hardware error -2:hardware busy, try again later -3: Parameter error 990x:Extended delay where x is a number 0-5

R1–7.3.9.2–2. For the EXTI2C option: The Buffer Pointer must point to a contiguous real storage area large enough to contain the I^2C command and any associated data (maximum of 261 bytes).

R1–7.3.9.2–3. For the EXTI2C option: The Buffer format for the write operation must be as defined in Table 65, “EXTI2C Buffer Write Operation Format,” on page 167.

Table 65. EXTI2C Buffer Write Operation Format

Condition	Byte #	Content
On Call	0	0x00
	1-3	Address of I2C resource
	4	Length of op. (1-255 with 0 specifying 256)
	5...	Data
On Return - I2C OK	0	Buffer unmodified
	1-3	
	4	
	5...	

Table 65. EXTI2C Buffer Write Operation Format ***(Continued)***

Condition	Byte #	Content
On Return - I2C error	0	0x01
	1-3	Address of I2C resource
	4	I2C operation error code

Firmware and Software Implementation Note: When the *ibm,exti2c* RTAS call write operation returns after the operation has been enqueued by the firmware but prior to completion by the hardware (therefore the operation status is truly not known), the *ibm,exti2c* RTAS call can return a *Status* of 0 (success) with the buffer unmodified.

R1–7.3.9.2–4. For the EXTI2C option: The Buffer format for a read operation, if supported, must be as defined in Table 66, “EXTI2C Buffer Read Operation Format (Optional),” on page 168.

Table 66. EXTI2C Buffer Read Operation Format (Optional)

Condition	Byte #	Content
On Call	0	0x80
	1-3	Address of I2C resource
	4	Length of op. (1-255 with 0 specifying 256)
	5...	Data
On Return - I2C OK	0	0x80
	1-3	Address of I2C resource
	4	Length of op. (1-255 with 0 specifying 256)
	5...	Data
On Return - I2C error	0	0x81
	1-3	Address of I2C resource
	4	I2C operation error code

R1–7.3.9.2–5. For the EXTI2C option: If read operations are not supported and a read operation is attempted, then the platform must return a *Status* of -3.

R1–7.3.9.2–6. For the EXTI2C option: The maximum total Extended Delay imposed by the *ibm,exti2c* command for a single I^2C operation must be less than 2 seconds.

R1–7.3.9.2–7. For the EXTI2C option: When the *ibm,exti2c* RTAS call returns an EXTI2C buffer containing an I^2C operation error code, the RTAS *Status* code must be 0 (success).

Firmware and Software Implementation Note: When the *ibm,exti2c* RTAS call returns after the operation has been enqueued by the firmware but prior to completion by the hardware (therefore the operation status is truly not known), the *ibm,exti2c* RTAS call can return a *Status* of 0 (success) with the buffer unmodified.

7.3.10 PowerPC External Interrupt Option

The RTAS calls used to access the facilities of the PowerPC External Interrupt option need not be serialized by the calling OS. Other RTAS rules such as being called in real mode with interrupts disabled still apply.

Note: These RTAS calls make the PowerPC External Interrupt option Logical Partition (LPAR) ready.

7.3.10.1 ibm,get-xive

R1–7.3.10.1–1. For the PowerPC External Interrupt option: RTAS must implement an *ibm,get-xive* call using the argument call buffer defined by Table 67, “ibm,get-xive Argument Call Buffer,” on page 169

R1–7.3.10.1–2. For the PowerPC External Interrupt option: The *ibm,get-xive* call must be reentrant to the number of processors on the platform.

R1–7.3.10.1–3. For the PowerPC External Interrupt option: The *ibm,get-xive* argument call buffer for each simultaneous call must be physically unique.

R1–7.3.10.1–4. For the PowerPC External Interrupt option: The *ibm,get-xive* call must return the current values of the server number and priority fields, as set by the last *ibm,set-xive* call (priority initialized to least favored level by firmware at boot), of the External Interrupt Vector Entry associated with the interrupt number provided as an input argument unless prevented by Requirement R1–14.10–22.

R1–7.3.10.1–5. For the PowerPC External Interrupt option: The *ibm,get-xive* call must return the *Status* of -3 (Argument Error) for an unimplemented Interrupt # (not reported via an **`“interrupt-ranges”`** property).

R1–7.3.10.1–6. For the PowerPC External Interrupt option combined with the Platform Reserved Interrupt Priority Level option: The *ibm,get-xive* call must return the *Status* of -3 (Argument Error) for an platform reserved interrupt priority (reported via an the **`“ibm,plat-res-int-priorities”`** property).

Table 67. *ibm,get-xive* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,get-xive*
	Number Inputs	1
	Number Outputs	3
	Interrupt #	From **`“interrupt-ranges”`** property
Out	*Status*	0: Success -1: Hardware Error -3: Argument Error (Optional)
	Server #	0x0 - 2**`“ibm,interrupt-server#-size”`**
	Priority	0x0 - 0xff

7.3.10.2 ibm,set-xive

R1–7.3.10.2–1. For the PowerPC External Interrupt option: RTAS must implement an *ibm,set-xive* call using the argument call buffer defined by Table 68, “ibm,set-xive Argument Call Buffer,” on page 170

R1–7.3.10.2–2. For the PowerPC External Interrupt option: The *ibm,set-xive* call must be reentrant to the number of processors on the platform.

R1–7.3.10.2–3. For the PowerPC External Interrupt option: The *ibm,set-xive* argument call buffer for each simultaneous call must be physically unique.

R1–7.3.10.2–4. For the PowerPC External Interrupt option: The *ibm,set-xive* call must set values of the server number and priority fields of the External Interrupt Vector Entry (XIVE) and/or firmware saved priority value (if the interrupt source controller does not use an interrupt Enable Register and the interrupt source is masked off, either due to a previous *ibm,int-off* call or because the interrupt source was never enabled with an *ibm,int-on* call since boot), associated with the interrupt number provided as an input argument unless prevented by Requirement R1–14.10–20.

R1–7.3.10.2–5. For the PowerPC External Interrupt option: The *ibm,set-xive* call must return the *Status* of -3 (Argument Error) for an unimplemented Interrupt number.

R1–7.3.10.2–6. For the PowerPC External Interrupt plus the Platform Reserved Interrupt Priority Level option: The *ibm,set-xive* call must return the *Status* of -3 (Argument Error) for an reserved Priority value (as reported via an **`“ibm,plat-res-int-priorities”`** property).

Table 68. *ibm,set-xive* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,set-xive*
	Number Inputs	3
	Number Outputs	1
	Interrupt #	Interrupt number from appropriate OF device tree property
	Server #	0x00 - 2**`“ibm,interrupt-server#-size”`**
	Priority	0x00 - 0xFF
Out	*Status*	0: Success -1: Hardware Error -3 Argument Error (Optional)

7.3.10.3 ibm,int-off

R1–7.3.10.3–1. For the PowerPC External Interrupt option: RTAS must implement an *ibm,int-off* call using the argument call buffer defined by Table 69, “ibm,int-off Argument Call Buffer,” on page 171

R1–7.3.10.3–2. For the PowerPC External Interrupt option: The *ibm,int-off* call must be reentrant to the number of processors on the platform.

R1–7.3.10.3–3. For the PowerPC External Interrupt option: The *ibm,int-off* argument call buffer for each simultaneous call must be physically unique.

R1–7.3.10.3–4. For the PowerPC External Interrupt option: The *ibm,int-off* call must disable interrupts from the interrupt source associated with the interrupt number provided as an input argument unless prevented by Requirement R1–14.10–24.

R1–7.3.10.3–5. For the PowerPC External Interrupt option: If the interrupt source controller uses an Interrupt Enable Register, the *ibm,int-off* call must reset the mask bit associated with the specified interrupt number; or if the interrupt source controller does not use an interrupt Enable Register, the *ibm,int-off* call must save the priority value of the XIVE for later restoration by the *ibm,int-on* call, or presentation by the *ibm,get-xive* call and set the priority value of the XIVE to the least favored priority value (0xFF), unless prevented by Requirement R1–14.10–24.

R1–7.3.10.3–6. For the PowerPC External Interrupt option: The *ibm,int-off* call must return the *Status* of -3 (Argument Error) for an unimplemented Interrupt number.

Table 69. *ibm,int-off* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,int-off*
	Number Inputs	1
	Number Outputs	1
	Interrupt #	Interrupt number from appropriate OF device tree property
Out	*Status*	0: Success -1: Hardware Error -3 Argument Error (Optional)

7.3.10.4 ibm,int-on

R1–7.3.10.4–1. For the PowerPC External Interrupt option: RTAS must implement an *ibm,int-on* call using the argument call buffer defined by Table 70, “ibm,int-on Argument Call Buffer,” on page 172.

R1–7.3.10.4–2. For the PowerPC External Interrupt option: The *ibm,int-on* call must be reentrant to the number of processors on the platform.

R1–7.3.10.4–3. For the PowerPC External Interrupt option: The *ibm,int-on* argument call buffer for each simultaneous call must be physically unique.

R1–7.3.10.4–4. For the PowerPC External Interrupt option: The *ibm,int-on* call must enable interrupts from the interrupt source associated with the interrupt number provided as an input argument unless prevented by Requirement R1–14.10–23.

R1–7.3.10.4–5. For the PowerPC External Interrupt option: If the interrupt source controller uses an Interrupt Enable Register, the *ibm,int-on* call must set the mask bit associated with the specified interrupt number; or if the interrupt source controller does not use an interrupt Enable Register, the *ibm,int-on* call must restore the XIVE priority value saved by the previous *ibm,int-off* call (initialized by the firmware to the least favored level at boot) unless prevented by Requirement R1–14.10–24.

R1–7.3.10.4–6. For the PowerPC External Interrupt option: The *ibm,int-on* call must return the *Status* of -3 (Argument Error) for an unimplemented Interrupt number.

Table 70. *ibm,int-on* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,int-on*
	Number Inputs	1
	Number Outputs	1
	Interrupt #	Interrupt number from appropriate OF device tree property
Out	*Status*	0: Success -1: Hardware Error -3 Argument Error (Optional)

7.3.10.5 MSI Support

This section describes the RTAS calls required when the MSI option is implemented. See Section 6.2.3, “MSI Option,” on page 103 for other platform requirements for the MSI option.

The Message Signaled Interrupt (MSI) and Enhanced MSI (MSI-X) capability of PCI IOAs in many cases allows for greater flexibility in assignment of external interrupts to IOAs than the predecessor Level Sensitive Interrupt (LSI) capability, and in some cases allows the treatment of MSIs[1] as a resource pool that is reassigned based on availability of MSIs and the need of an IOA function for more interrupts than initially assigned. Platforms that implement the MSI option implement the *ibm,change-msi* and *ibm,query-interrupt-source-number* RTAS calls.

7.3.10.5.1 ibm,change-msi

The OS uses the *ibm,change-msi* RTAS call to query the initial number of MSIs assigned to a PCI configuration address (that is, to an IOA function) and to request a change in the number of MSIs assigned, when the platform allows for dynamic reassignment of MSIs for the IOA function. The *ibm,change-msi* RTAS call allows the caller to allow the platform to select MSI or MSI-X, to specifically select MSI or MSI-X or, if LSIs are allocated by the firmware for the IOA function, to change to LSI (by removal of the MSIs assigned). The interrupt source numbers assigned to an IOA function are queried via the *ibm,query-interrupt-source-number* RTAS call. The *ibm,query-interrupt-source-number* RTAS call is called iteratively, once for each interrupt assigned to the IOA function. The interrupt source numbers returned by the *ibm,query-interrupt-source-number* RTAS call are the numbers used to control the interrupt as in the *ibm,get-xive*, *ibm,set-xive*, *ibm,int-on*, and *ibm,int-off* RTAS calls.

If a device driver is willing to live with the platform-assigned initial number of MSIs, then the device driver does not need to use the *ibm,change-msi* RTAS call, and can instead use the *ibm,query-interrupt-source-number* RTAS call to determine the number of interrupts assigned to each IOA function.

An OS may abandon the effort to change the MSIs for a given configuration address after the first call to *ibm,change-msi* and prior to a call which gets a status back indicating completion, by calling again with the same PCI configuration address but with a *Function* number of 2 (set to default number of interrupts) and a *Sequence Number* of 1. RTAS never returns a *Status* of -2 or 990x when the call is made with a *Function* number of 2.

If an OS successfully changes the number of interrupts, then it should consider removing the increase when it deconfigures the IOA function, especially if it starts with zero and wants to be backward compatible with older device drivers that may not understand MSIs. To remove all MSIs, set the Requested Number of Interrupts to zero. But it should be noted, that once set to zero, there is no guarantee that on a future request there will be any MSIs available to assign

1.This architecture will refer generically to the MSI and MSI-X capabilities as simply “MSI,” except where differentiation is required.

from the pool. Adding MSIs to an IOA function which has LSIs assigned disables those LSIs but does not remove them, and then removing the MSIs that replaced the LSIs re-uses the same (previously removed) LSIs (mapped to the same LSI source numbers as the previous LSI source numbers).

The presence of the **"ibm,change-msix-capable"** property specifies that the platform implements the version of this RTAS call that allows *Number Outputs* equal to 4 and *Functions* 3 and 4.

If the *ibm,change-msi* RTAS call is made with *Number Outputs* equal to 4 or with *Function* equal to 3 or 4 when the **"ibm,change-msix-capable"** property does not exist in the **/rtas** node, then the call will return a *Status* of -3 (Invalid Parameter). Specifying *Function* 3 (MSI) also disables MSI-X for the specified IOA function, and likewise specifying *Function* 4 (MSI-X) disables MSI for the IOA function. It is unnecessary to specify a *Requested Number of Interrupts* of zero when switching between MSI and MSI-X. Specifying the *Requested Number of Interrupts* to zero for either *Function* 3 or 4 removes all MSI & MSI-X interrupts from the IOA function. It is permissible to use LSI, MSI and MSI-X on different IOA functions.

The default (initial) assignment of interrupts is defined in Section 6.2.3, "MSI Option," on page 103.

R1–7.3.10.5.1–1. For the MSI option: The platform must implement the *ibm,change-msi* call using the argument call buffer defined by Table 71, "ibm,change-msi Argument Call Buffer," on page 173.

Table 71. *ibm,change-msi* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,change-msi.*
	Number Inputs	6
	Number Outputs	3 or 4, when the **"ibm,change-msix-capable"** property is present, 3 otherwise.
	Config_addr	Configuration Space Address (Register field set to 0)
	PHB_Unit_ID_Hi	Represents the most-significant 32-bits of the Unit ID of the PHB that corresponds to the *config_addr*
	PHB_Unit_ID_Low	Represents the least-significant 32-bits of the Unit ID of the PHB that corresponds to the *config_addr*
	Function	Determines action of this call: 0: Query only (only return actual number of MSI or MSI-X interrupts assigned to the PCI configuration address). 1: If *Number Outputs* is equal to 3, request to set to a new number of MSIs (including set to 0). If the **"ibm,change-msix-capable"** property exists and *Number Outputs* is equal to 4, request is to set to a new number of MSI or MSI-X (platform choice) interrupts (including set to 0). 2: Request to set back to the default number of interrupts (also aborts a change in progress; that is, one that has previously returned a *Status* of -2 or 990x) 3: (Only valid if **"ibm,change-msix-capable"** exists): Request to set to a new number of MSIs (including set to 0) 4: (Only valid if **"ibm,change-msix-capable"** exists): Request to set to a new number of MSI-X interrupts (including set to 0)
	Requested Number of Interrupts	The total number of MSIs being requested for the PCI configuration address. A value of 0 is specified in order to remove all MSIs for the PCI configuration address. This input parameter is ignored by RTAS for *Function* values other than 1, 3, or 4.
	Sequence Number	Integer representing the sequence number of the call. First call in sequence starts with 1, following calls (if necessary) use the *Next Sequence Number* returned from the previous call.

Table 71. *ibm,change-msi* Argument Call Buffer ***(Continued)***

Parameter Type	Name	Values
Out	*Status*	-3: Parameter error -2: Call again -1: Hardware error 0: Success 990x: Extended Delay
	Final Number of Interrupts	Number of interrupts assigned to the PCI configuration address at the successful completion of this call (*Status* of 0). For *Function* 1, 3, or 4, if a greater number was requested than what was previously assigned, the final number may be less than what was requested, even though a *Status* of 0 is returned.
	Next Sequence Number	Integer to be returned as the *Sequence Number* parameter on the next call. This output is only valid if a *Status* of -2 or 990x is returned.
	Type of Interrupts	This field is only valid when the *Final Number of Interrupts* is non-zero. 1: MSI 2: MSI-X

R1–7.3.10.5.1–2. For the MSI option: The *Final number of Interrupts* and *Type of Interrupts* must be valid when the platform returns a *Status* of 0 (Success), regardless of whether the original number and final number of interrupts assigned is different and regardless of whether or not the platform allows MSI resources to be reassigned for the specified PCI configuration address.

R1–7.3.10.5.1–3. For the MSI option: The platform must return a *Status* of -3 (Parameter error) from *ibm,change-msi*, with no change in interrupt assignments if the PCI configuration address does not support MSI and *Function* 3 was requested (that is, the **`"ibm,req#msi"`** property must exist for the PCI configuration address in order to use *Function* 3), or does not support MSI-X and *Function* 4 is requested (that is, the **`"ibm,req#msi-x"`** property must exist for the PCI configuration address in order to use *Function* 4), or if neither MSIs nor MSI-Xs are supported and *Function* 1 is requested.

R1–7.3.10.5.1–4. For the MSI option: If there are zero MSIs assigned to the target IOA function but there is one or more LSIs assigned, then a call to *ibm,change-msi* which successfully changes the number of MSIs assigned to non-zero must also disable the LSIs in the IOA function's configuration space and must keep the LSI platform resources available to the IOA function in the case the MSIs are removed (see Requirement R1–7.3.10.5.1–5).

R1–7.3.10.5.1–5. For the MSI option: If there are a non-zero number of MSIs assigned to the target IOA function and if that IOA function originally had some LSIs assigned, then a call to *ibm,change-msi* which successfully changes the number of MSIs assigned to zero must also reassign any LSIs that were originally assigned to that IOA function, using the same interrupt number that was originally assigned (that is, the platform must reserve an originally assigned LSI for a IOA function in case it needs to reassign it), and must enable the LSIs in the IOA function's configuration space.

R1–7.3.10.5.1–6. For the MSI option: If the platform supports the changing of MSIs, then it must support the reduction in the number of interrupts by the *ibm,change-msi* call, including setting the number of MSIs to 0.

R1–7.3.10.5.1–7. For the MSI option: On the first call of *ibm,change-msi*, the *Sequence Number* must be a 1.

R1–7.3.10.5.1–8. For the MSI option: If *ibm,change-msi* returns a *Status* of -2 (Call again) or 990x (Extended Delay), then the caller must provide on the next call to *ibm,change-msi*, one of the following:

- All input parameters the same as the initial call except with the *Sequence Number* set to the value in *Next Sequence Number* returned from the previous *ibm,change-msi* call.
- All input parameters the same as the initial call except with *Function* set to 2 and the *Sequence Number* set to 1, if the caller is wanting to abort the previously started *ibm,change-msi* operation.

R1–7.3.10.5.1–9. For the MSI option: If the *ibm,change-msi* RTAS call returns something other than 0 for the *Final Number of Interrupts*, then the *ibm,query-interrupt-source-number* RTAS call must be used to get the current interrupt source numbers, even if the *ibm,change-msi* call has returned the same number of interrupts as before the call.

R1–7.3.10.5.1–10. For the MSI option: Firmware must not return a *Status* of -2 or 990x when the *Requested Number of Interrupts* is set to 0 or for *Function* 0 (query only) or for *Function* 2 (set back to default number).

R1–7.3.10.5.1–11. For the MSI option: When the *set-indicator* RTAS call is made to isolate an IOA (for both DLPAR and PCI Hot Plug operations), the platform must release any additional MSI numbers that were obtained through the *ibm,change-msi* RTAS call and make them available for use by other *ibm,change-msi* calls.

R1–7.3.10.5.1–12. For the MSI option: An OS or device driver that is calling *ibm,change-msi* for the purpose of changing the number or type of interrupts for the IOA function must assure that the IOA function cannot be actively performing operations that will generate interrupts during the process of changing the number or type of interrupts.

R1–7.3.10.5.1–13. For the MSI option: The platform must restore the IOA's MSI configuration space after a reset operation which occurs following boot, to what it was previous to the reset operation, and provide the same MSI assignments through the reset operation, unless a DR isolate/unisolate operation has been performed (in which case the IOA's MSI configuration space is set as it would at boot time).

Software Implementation Notes:

1. Interrupt source numbers for MSIs are not necessarily be assigned contiguously.
2. MSIs and MSI source numbers are not shared (see Requirement R1–6.2.3–5).
3. For a multi-function IOA, the *ibm,change-msi* call is called for each function for which the number of MSIs is to be changed.
4. When the 990x *Status* is returned, it is suggested that software delay for 10 raised to the x milliseconds (where x is the last digit of the 990x return code), before calling *ibm,change-msi* again. However, software may issue the *ibm,change-msi* call again either earlier or later than this.
5. During a sequence of calls which return -2 or 990x, software may abort at any time by setting the *Function* equal to 2 and the *Sequence Number* to 1.
6. When there is a non-zero number of MSI or MSI-X interrupts assigned, and when software attempts to change the type of interrupts (MSI to MSI-X interrupt or MSI-X to MSI) at the same time as changing the number of interrupts, the platform may return the same number of interrupts as previously assigned, even though a greater number is available. In this case a second call to *ibm,change-msi* to increase the number of interrupts may produce a greater number of interrupts.

7.3.10.5.2 ibm,query-interrupt-source-number

The *ibm,query-interrupt-source-number* RTAS call is used to query the interrupt source number and type (level sensitive for LSIs, edge triggered for MSIs) for a specific PCI IOA function's interrupt, if one exists. That is, for a given PCI configuration address (*PHB_Unit_ID_Hi, PHB_Unit_ID_Low,* and *config_addr*) and function interrupt number. This call is issued once for each interrupt of each IOA function, in order to obtain the interrupt source number and type for that interrupt. For example, if the *ibm,change-msi* RTAS call has previously returned a value of "n" interrupts for the IOA function, then the call is made "n" times for that function (with a relative interrupt number of 0 to n-1).

R1–7.3.10.5.2–1. For the MSI option: The platform must implement the *ibm,query-interrupt-source-number* RTAS call using the argument call buffer defined by Table 72, "ibm,query-interrupt-source-number Argument Call Buffer," on page 176.

Table 72. *ibm,query-interrupt-source-number* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,query-interrupt-source-number*
	Number Inputs	4
	Number Outputs	3
	Config_addr	Configuration Space Address (Register field set to 0)
	PHB_Unit_ID_Hi	Represents the most-significant 32-bits of the Unit ID of the PHB that corresponds to the *config_addr*
	PHB_Unit_ID_Low	Represents the least-significant 32-bits of the Unit ID of the PHB that corresponds to the *config_addr*
	IOA Function Interrupt Number	The relative number of the interrupt within the PCI configuration address, with a value of 0 being the first interrupt of the PCI configuration address.
Out	*Status*	-3: Parameter error -1: Hardware error 0: Success 1: No interrupt assigned for the given PCI configuration address and *IOA Function Interrupt Number.*
	Interrupt Source Number	The interrupt source number corresponding to the PCI configuration address and *IOA Function Interrupt Number*, when a *Status* of 0 is returned. Undefined for other *Status* values.
	Interrupt Source Trigger	The interrupt source trigger corresponding to the PCI configuration address and *IOA Function Interrupt Number*, when a *Status* of 0 is returned. Undefined for other *Status* values. 0: Level sensitive 1: Edge triggered

R1–7.3.10.5.2–2. For the MSI option: The interrupt source numbers returned by the *ibm,query-interrupt-source-number* RTAS call must be the numbers used to control the interrupt as in the *ibm,get-xive*, *ibm,set-xive*, *ibm,int-on*, and *ibm,int-off* RTAS calls.

R1–7.3.10.5.2–3. For the MSI option: The *ibm,query-interrupt-source-number* RTAS call must return a *Status* of 1 (no interrupt assigned) if the inputs specify a valid PCI configuration address and the PCI configuration address does not have an interrupt assigned for the specified *IOA Function Interrupt Number.*

Software and Firmware Implementation Note: Software may use the *ibm,query-interrupt-source-number* RTAS call for all *IOA Function Interrupt Number* values starting at 0 until a *Status* of 1 is returned, rather than using *ibm,change-msi Function* 0 (query). That is, the *ibm,query-interrupt-source-number* RTAS call works even when the **"ibm,req#msi"** property does not exist for the IOA (that is even when the IOA is not requesting one or more MSIs), This might be desirable, for example, if software never plans on using the capability to change the number of MSIs, and therefore does not have any other use for the *ibm,change-msi* call.

7.3.11 Enhanced I/O Error Handling (EEH) Option Functions

The EEH option requires several additional RTAS calls. In addition, the Error Injection option RTAS calls are required to be implemented, in order to be able to test device driver code that implements recovery based on the EEH option.

See also, Appendix H, “EEH Error Processing,” on page 835, for additional information about implementing EEH error recovery.

R1–7.3.11–1. For the EEH option: The IOA bus error injection function of the Error Injection option RTAS call must be implemented concurrently with the EEH option (that is, the *ioa-bus-error* token must exist in the **"ibm,errinjct-tokens"** property).

R1–7.3.11–2. For the EEH option: If the EEH option is implemented for the specified PE configuration address, then calls to the *ibm,set-eeh-option, ibm,set-slot-reset,* and *ibm,slot-error-detail* RTAS calls must be governed by Table 73, "PE State Transition Table," on page 178, otherwise if one of the invalid transitions in Table 73, "PE State Transition Table," on page 178 is attempted, then return a *Status* as defined by Table 74, "PE State Control," on page 179.

R1–7.3.11–3. If the EEH option is not implemented for the specified PE configuration address and a call is made to one of the *ibm,set-eeh-option, ibm,set-slot-reset,* or *ibm,slot-error-detail* RTAS calls, then return a *Status* of -3 (parameter error).

Software Implementation Note: Some transitions in Table 73, "PE State Transition Table," on page 178 are made asynchronously to the OS by the platform (in exceptional cases; see table for details). If software receives a *Status* of -7 (Unexpected state change) on an RTAS call which is attempting to change state in Table 73, "PE State Transition Table," on page 178, then software should read the state again via the *ibm,read-slot-reset-state2* RTAS call, in order to obtain the current state. Some legacy implementations may return a -1 instead of a -7.

R1–7.3.11–4. For the EEH option: If the platform activates the reset to a PE (for example, as part of a recovery action above the PE), including the case where the platform has temporarily deactivated and then reactivated the reset, then the platform must hide such PE state transition(s) from the OS by returning a *Status* of 5 (PE is unavailable) with *PE Unavailable Info* indicating a non-zero value (temporarily unavailable) for the *ibm,read-slot-reset-state2* RTAS call, until which time the required minimum reset active hold time for the hardware within the PE has been met.

Software Implementation Note: Relative to the platform automatically resetting the PE as part of error recovery, as mentioned in Requirement R1–7.3.11–4, the *PE Recovery Info* output of the *ibm,read-slot-reset-state2* RTAS call is provided to enable the software to determine that such a reset has occurred.

R1–7.3.11–5. For the EEH option: If the platform deactivates the reset to a PE, except in the case where the OS has instructed it to do so with the *ibm,set-slot-reset Function* 0, then the platform must do all the following:

a. Hide such a deactivation from the OS during the time that the PE reset is deactivated by returning a *Status* of 5 (PE is unavailable) with *PE Unavailable Info* indicating a non-zero value (temporarily unavailable) for the *ibm,read-slot-reset-state2* RTAS call.

b. Force OS MMIO accesses to the PE during the deactivation time to look like the PE is reset.

c. Prevent the PE from introducing errors into the system (for example, from DMA or due to the reset being deactivated prior to the proper active hold time).

d. Reactivate the reset, hiding the reset active hold time as required by Requirement R1–7.3.11–4, or force the PE into the permanently unavailable state (return a *Status* of 5 (PE is unavailable) with *PE Unavailable Info* indicating a zero value for the *ibm,read-slot-reset-state2* RTAS call).

R1–7.3.11–6. For the EEH option: The Bridged-I/O EEH option must be implemented concurrently with the EEH option.

R1–7.3.11–7. For the EEH option: The 64 bit IOA bus error injection function of the Error Injection option RTAS call must be implemented concurrently with the EEH option (that is, the *ioa-bus-error-64* token must exist in the **`"ibm,errinjct-tokens"`** property).

R1–7.3.11–8. For the EEH option: The platform must implement the *ibm,configure-pe* RTAS call.

Table 73. PE State Transition Table

Initial PE State[a]	Final PE State[a]					
	0 Not Reset *Load*/*Store* Allowed[b] DMA Allowed[c] (Normal Operations)	1 Reset[d] *Load*/*Store* Disabled[e] DMA Disabled[f]	2 Not Reset[g] *Load*/*Store* Disabled[h] DMA Disabled[i]	4 Not Reset *Load*/*Store* Allowed[b] DMA Disabled[i]	5 Temporarily Unavailable[j]	5 Permanently Unavailable[k]
0 Not Reset *Load*/*Store* Allowed[b] DMA Allowed[c] (Normal Operations)		*ibm,set-slot-reset Function* 1 or 3, or via a hardware initiated action[l]	Hardware causes this state transition when EEH is enabled and an error occurs or firmware may cause due to higher level error recovery action	Not a valid transition	Platform initiated action	*ibm,slot-error-detail Function* 2, or platform detected permanent error
1 Reset *Load*/*Store* Disabled DMA Disabled[f]	*ibm,set-slot-reset Function* 0[m]	*ibm,set-slot-reset Function* 1 or 3	Not a valid transition	Not a valid transition	Platform initiated action	*ibm,slot-error-detail Function* 2, or platform detected permanent error
2 Not Reset *Load*/*Store* Disabled[e] DMA Disabled[i]	Not a valid transition Must go through state 4 or state 1	*ibm,set-slot-reset Function* 1 or 3, or via a hardware initiated action[l]		*ibm,set-eeh-option Function* 2	Platform initiated action	*ibm,slot-error-detail Function* 2, or hardware detected permanent error
4 Not Reset *Load*/*Store* Allowed[b] DMA Disabled[i]	*ibm,set-eeh-option Function* 3 See Requirement R1–7.3.11.1–4	*ibm,set-slot-reset Function* 1 or 3, or via a hardware initiated action[l]	Hardware causes this state transition when EEH is enabled and an error occurs or firmware may cause due to higher level error recovery action		Platform initiated action	*ibm,slot-error-detail Function* 2, or platform detected permanent error
5 Temporarily Unavailable[j]	Not a valid transition	Platform initiated action	Platform initiated action[n]	Not a valid transition		*ibm,slot-error-detail Function* 2, or platform detected permanent error
5 Permanently Unavailable[k]	Power cycle, Partition reboot, or DLPAR re-assignment	Not a valid transition	Not a valid transition	Not a valid transition	Not a valid transition	

a. The state as would be returned from *ibm,read-slot-reset-state2*, when no asynchronous platform transition has occurred.
b. *Load*/*Store* allowed means the *Loads* or *Stores* channel is open to the PE, and not necessarily that the PE itself has its MMIO space enabled. The components within the PE also contain enable/disable bits (for example, the PCI configuration space Memory Space and IO Space enable bits in the PCI header Device Control register).
c. DMA allowed means the DMA channel is open to the PE, if the PE itself has its DMA enabled. The components within the PE also contain enable/disable bits (for example, the PCI configuration space Bus Master enable bit in the PCI header Device Control register).
d. Reset may mean that the PE is being held in the reset state by a reset signal or Hot Reset (PCI Express), or that it may have been put into this state by the platform via a PCI Express Function Level Reset (FLR) in response to the *ibm,set-slot-reset* RTAS call. In the case of FLR, the platform makes the pulse of the FLR look like the Hot Reset case to the OS in terms of the Reset state (See Section 7.3.11.2, "ibm,set-slot-reset," on page 182 for more information). Note that the platform does not monitor writes to the FLR bit of an IOA, and so OS or Device Driver writes directly to the FLR bit on an IOA will not affect the PE State as shown in Table 73.
e. *Load*/*Store* disabled means that either the PE is in the MMIO Stopped state or that the PE is reset, the latter giving the appearance of being MMIO Stopped.
f. DMA disabled means that either the PE is in the DMA Stopped state or that the PE is reset, the latter giving the appearance of being DMA Stopped.
g. Although the current state is "Not Reset", the PE may have been reset by the platform in the process of getting to this state. The *PE Recovery Info* output of the *ibm,read-slot-reset-state2* RTAS call will indicate if the platform has done such a reset.
h. In the MMIO Stopped state.
i. In the DMA Stopped state.
j. Temporarily unavailable is signaled by a non-zero value returned in the *PE Unavailable Info* of the *ibm,read-slot-reset-state2* RTAS calls.
k. Permanently unavailable is signaled by a zero value returned in the *PE Unavailable Info* of the *ibm,read-slot-reset-state2* RTAS calls.

l. The hardware would not normally initiate this transition; such implementation would only exist where the platform only implements EEH Stopped State via a reset (always) of the PE (that is, only implements states 0, 1, and 5 of this table), which is not applicable for IBM LoPAPR Compliant platforms.
m. The platform also removes the PE from the EEH Stopped state, if applicable, on the transition from state 1 to state 0.
n. This transition cannot occur if the PE was in the reset state prior to the platform transition to the temporarily unavailable state. See Requirement R1–7.3.11–5.

Table 73, “PE State Transition Table,” on page 178 depicts the four main functions for controlling a PE’s state:

- *ibm,set-eeh-option Function* 2 for releasing a PE from the MMIO Stopped State, when the PE is in State 2 (MMIO Stopped State and DMA Stopped State both active).
- *ibm,set-eeh-option Function* 3 for releasing a PE from the DMA Stopped State, when the PE is in State 3 (MMIO Stopped State not active and DMA Stopped State active).
- *ibm,set-slot-reset Function* 0 for releasing a PE’s reset.
- *ibm,set-slot-reset Function* 1for activating a PE’s reset.

Implementation Note: In the last two bullets, above, for the case of the platform’s use of FLR for resetting the PE, the meaning of “activating” and “deactivating” the PE’s reset has slightly different meaning, but the platform makes the EEH recovery model transparent. See Section 7.3.11.2, “ibm,set-slot-reset,” on page 182 for more details.

Table 74, “PE State Control,” on page 179 is a summary of the expected results of the above four operations. The -7 *Status* returns generally will occur for the cases where the PE state has been changed asynchronously to the OS by the platform. In these cases, software should read the state again (via the *ibm,read-slot-reset-state2* RTAS call) in order to determine the current hardware state.

Table 74. PE State Control

RTAS Call	Function	Result and *Status*	**Initial PE State**[a]					
			0 Not Reset *Load*/*Store* Allowed[b] DMA Allowed[c] (Normal Operations)	1 Reset[d] *Load*/*Store* Disabled[e] DMA Disabled[f]	2 Not Reset *Load*/*Store* Disabled[g] DMA Disabled	4 Not Reset *Load*/*Store* Allowed[b] DMA Disabled[h]	5 Temporarily Unavailable[i]	5 Permanently Unavailable[j]
ibm,set-eeh-option	*Function* 2 Release PE for *Load*/*Store*	Result	no-op	no-op	Transition from state 2 to 4	no-op	no-op	no-op
		Status[k]	-3	-7	0	-3	-7	-7
	Function 3 Release PE for DMA	Result	no-op	no-op	no-op	Transition from state 4 to 0	no-op	no-op
		Status[k]	-3	-7	-7	0	-7	-7
ibm,set-slot-reset	*Function* 0 Deactivate the reset to the PE[l]	Result	no-op	Transition from state 1 to 0[m]	no-op	no-op	no-op	no-op
		Status[k]	0	0	-7	-3	-7	-7
	Function 1 or 3 Activate the reset to the PE	Result[n]	Transition from state 0 to 1	no-op	Transition from state 2 to 1	Transition from state 4 to 1	no-op	no-op
		Status[k, n]	0	0	0	0	-7	-7

a. The state as would be returned from *ibm,read-slot-reset-state2*, when no asynchronous platform transition has occurred.
b. *Load*/*Store* allowed means the *Loads* or *Stores* channel is open to the PE, and not necessarily that the PE itself has its MMIO space enabled. The components within the PE also contain enable/disable bits (for example, the PCI configuration space Memory Space and IO Space enable bits in the PCI header Device Control register).

c. DMA allowed means the DMA channel is open to the PE, if the PE itself has its DMA enabled. The components within the PE also contain enable/disable bits (for example, the PCI configuration space Bus Master enable bit in the PCI header Device Control register).
d. Reset may mean that the PE is being held in the reset state by a reset signal or Hot Reset (PCI Express), or that it may have been put into this state by the platform via a PCI Express Function Level Reset (FLR) in response to the *ibm,set-slot-reset* RTAS call. In the case of FLR, the platform makes the pulse of the FLR look like the Hot Reset case to the OS in terms of the Reset state (See Section 7.3.11.2, "ibm,set-slot-reset," on page 182 for more information). Note that the platform does not monitor writes to the FLR bit of an IOA, and so OS or Device Driver writes directly to the FLR bit on an IOA will not affect the PE State as shown in Table 73.
e. *Load*/*Store* disabled means that either the PE is in the MMIO Stopped state or that the PE is reset, the latter giving the appearance of being MMIO Stopped.
f. DMA disabled means that either the PE is in the DMA Stopped state or that the PE is reset, the latter giving the appearance of being DMA Stopped.
g. In the MMIO Stopped state.
h. In the DMA Stopped state.
i. Temporarily unavailable is signaled by a non-zero value returned in the *PE Unavailable Info* of the *ibm,read-slot-reset-state2* RTAS calls.
j. Permanently unavailable is signaled by a zero value returned in the *PE Unavailable Info* of the *ibm,read-slot-reset-state2* RTAS calls.
k. A *Status* of -3 is returned instead of 0 or -7 if an invalid PCI configuration address is used. An invalid PCI configuration address is generally one which is not a PE address or which is not assigned to the OS. However, some platforms may allow resetting within the PE or outside the PE, providing this does not violate other requirements defined by this architecture. Also, some legacy implementations may return a -1 or -3 instead of a -7, but all implementations are required to implement the -7 *Status*, where appropriate.
l. In the case of the use of FLR by the platform to reset the PE, the "activate" and "deactivate" of the reset has a different meaning than for the Hot Reset case. For FLR, the platform makes the pulse of the FLR look like the Hot Reset case to the OS in terms of the Reset state (See Section 7.3.11.2, "ibm,set-slot-reset," on page 182 for more information).
m. The platform also removes the PE from the EEH Stopped state, if applicable, on the transition from state 1 to state 0.
n. For *Function* 3, if *Function* 3 is not implemented, then a *Status* of -3 is returned. For *Function* 3, if implemented in the RTAS call, but not implemented for the specified PCI configuration address, then a *Status* of -8 is returned. In either of these cases, the PE state is not changed. If *Function* 3 is implemented, then the platform indicates this by the **`"ibm,reset-capabilities"`** property in the OF device tree.

The PE configuration address (*PHB_Unit_ID_Hi, PHB_Unit_ID_Low, and config_addr*) for the domain is the PCI configuration address for the PE primary bus and is the same format as used for the *ibm,read-pci-config* and *ibm,write-pci-config* calls (see Requirement R1–7.3.4–1), except that the Register field is set to 0. The PE configuration address is obtained as indicated in Table 9, "Conventional PCI Express PE Support Summary," on page 72.

7.3.11.1 *ibm,set-eeh-option*

This call is used to enable and disable the EEH domain of a PE, to remove a PE from the MMIO Stopped state to continue *Load* and *Store* operations to the domain, and to remove a PE from the DMA Stopped state to continue DMA operations to the domain. The PE configuration address (*PHB_Unit_ID_Hi, PHB_Unit_ID_Low, and config_addr*) for the PE is obtained as defined in Section 7.3.11, "Enhanced I/O Error Handling (EEH) Option Functions," on page 176.

R1–7.3.11.1–1. For the EEH option: RTAS must implement an *ibm,set-eeh-option* call using the argument call buffer defined by Table 75, "ibm,set-eeh-option Argument Call Buffer," on page 181.

Table 75. *ibm,set-eeh-option* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,set-eeh-option*
	Number Inputs	4
	Number Outputs	1
	Config_addr	PE configuration address (Register fields set to 0)
	PHB_Unit_ID_Hi	Represents the most-significant 32-bits of the Unit ID of the PHB that corresponds to the *config_addr*
	PHB_Unit_ID_Low	Represents the least-significant 32-bits of the Unit ID of the PHB that corresponds to the *config_addr*
	Function	0: Disable EEH option for the PE (no-op for PEs with PCI Express IOAs) 1: Enable EEH option for the PE 2: Release the PE for *Load*/*Store* operations 3: Release the PE for DMA operations 4: Enable EEH io-event interrupt for this PE 5: Disable EEH io-event interrupt for this PE
Out	*Status*	0: Success -1: Hardware Error -3: Parameter Error -7 Unexpected state change

Software and Platform Implementation Note: For platforms that enable EEH by default, *ibm,set-eeh-option Function* 0 (disable EEH) is a no-op. However, *ibm,set-eeh-option Function* 1 (enable EEH) is still required as a signalling method from the device driver to the platform that the device driver is at least EEH aware (see Requirement R1–4.4.1.1–18).

R1–7.3.11.1–2. For the EEH option: Software must use the *ibm,get-config-addr-info2* RTAS call, when supported, to get the EEH domain span of the PE, otherwise software must use the *ibm,read-slot-reset-state2* RTAS call in order to determine the span, and then software should attempt to perform all *ibm,set-slot-reset* and *ibm,set-eeh-option* RTAS calls appropriately, based on the EEH capabilities and as governed by Table 73, “PE State Transition Table,” on page 178.

R1–7.3.11.1–3. For the EEH option: If the EEH option is implemented for the specified PE configuration address, on a call to the *ibm,set-eeh-option* with a *Function* of 0 (disable EEH) the platform must do one of the following:

- If any IOA in the PE is enabled (if any of the Bus Master, Memory Space or I/O Space bits in the Command Register of the IOA’s configuration space are 1), then do nothing and return a *Status* of 0 (Success).
- If the platform allows disabling of EEH and the disabling of EEH for the PE violates another requirement relative to LPAR, then the platform must not disable the EEH option for the specified PE configuration address and must return a -7 (unexpected state change) or a -1 (hardware error), with -7 preferred.
- If the platform allows disabling of EEH and the disabling of EEH for the PE does not violate the other requirements relative to LPAR, then clear the MMIO Stopped State and DMA Stopped State, and disable the EEH option, for the specified PE configuration address.
- If the default for the platform is EEH enabled, then do nothing and return a *Status* of 0 (Success).

R1–7.3.11.1–4. For the EEH option: If the OS allows the DMA to be enabled for a PE that is in the DMA Stopped state without the use of a reset operation (that is, the use of the *ibm,set-eeh-option* with a *Function* of 3), the

device driver must first do all necessary cleanup of its IOA to prevent the IOA from doing anything destructive when it starts DMA again.

R1–7.3.11.1–5. For the EEH option: If a device driver is going to enable EEH and the platform has not defaulted to EEH enabled, then it must do so before it does any operations with its IOA, including any configuration cycles or *Load* or *Store* operations.

R1–7.3.11.1–6. For the EEH option: If an EEH domain is enabled for a PE via the *ibm,set-eeh-option* RTAS call and if there are multiple IOAs or one or more multi-function IOAs in that PE, and if these functions are supported by multiple device drivers, then all of the device drivers for all the functions in that PE must be EEH enabled and be capable of coordinating EEH recovery procedures.

Software implementation Note: Protection against startup errors (configuration cycles, etc.), are every bit as important as protection against errors during normal operations. Although the quantity of operations is not as great, there is more of a chance of latent errors showing up during the startup phase.

R1–7.3.11.1–7. For the EEH option: If the Slot Level EEH Event Interrupt option is not implemented for the PE, then return a *Status* of -3 if *Function* 4 or 5 is attempted.

R1–7.3.11.1–8. For the EEH with the Slot Level EEH Event Interrupt option: *Function* 4 and 5 must be implemented for all PE under nodes that contain the **`"ibm,io-events-capable"`** property.

7.3.11.2 ibm,set-slot-reset

This call is used to reset a PE. The PE configuration address (*PHB_Unit_ID_Hi, PHB_Unit_ID_Low, and config_addr*) for the PE is obtained as defined in Section 7.3.11, “Enhanced I/O Error Handling (EEH) Option Functions,” on page 176. All PEs have the capability of being reset independently. Resets outside or within the PE are not architected, but may be allowed by the platform implementation, providing that it does not violate other requirements of this architecture.

The platform may use one of two methods to reset a PCI Express PE, when the *ibm,set-slot-reset* RTAS call is made with the *Function* 1/*Function* 0 (activate the reset/deactivate the reset).

- If the PE is a single function of a multi-function IOA, then the Function Level Reset (FLR) option is required to be implemented by the function, and the platform uses FLR to reset the function. When the platform uses FLR instead of Hot Reset to reset a PCI Express PE, the platform provides the **`"ibm,pe-reset-is-flr"`** property in the function’s OF Device Tree node, and provides the same EEH recovery model to the software, as in the Hot Reset case, and as defined by Table 73, “PE State Transition Table,” on page 178. The property is provided in the case where there may be slightly different device-specific reset recoveries by the software for the FLR case.

  Software Implementation Note: The platform does not monitor writes to the FLR bit of an IOA, and so OS or Device Driver writes directly to the FLR bit on an IOA will not affect the PE State as shown in Table 73, “PE State Transition Table,” on page 178.

- Otherwise, a PCI Express Hot Reset is used.

R1–7.3.11.2–1. For the EEH option: The *ibm,set-slot-reset* call must be implemented using the argument call buffer defined by Table 76, “ibm,set-slot-reset Argument Call Buffer,” on page 183.

Table 76. *ibm,set-slot-reset* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,set-slot-reset*
	Number Inputs	4
	Number Outputs	1
	Config_addr	PE configuration address (Register fields set to 0)
	PHB_Unit_ID_Hi	Represents the most-significant 32-bits of the Unit ID of the PHB that corresponds to the *config_addr*
	PHB_Unit_ID_Low	Represents the least-significant 32-bits of the Unit ID of the PHB that corresponds to the *config_addr*
	Function	0: Deactivate the reset to the PE 1: Activate the reset to the PE (for PCI Express, if the platform uses FLR to reset the PE, the platform provides the **"ibm,pe-reset-is-flr"** property in the function's OF Device Tree node) 3: (optional) Activate the reset to the PE, using a PCI Express Fundamental Reset
Out	*Status*	0: Success -1: Hardware Error -3: Parameter Error -7: Unexpected state change -8: Fundamental Reset not defined for this configuration address

R1–7.3.11.2–2. For the EEH option: After activation of the reset (*Function* 1 or 3), software must delay the deactivation of the reset (*Function* 0) to that PE via the *ibm,set-slot-reset* call, until the minimum reset signal active time has elapsed, as designated by the bus specifications for the particular type bus or buses involved (100 millisecond for PCI).

Software Implementation Notes:

1. The device driver is responsible for any additional clean up required beyond that provided by a reset to the IOA. For PCI Express, the clean up may be slightly different based on whether the platform used FLR or Hot Reset to reset the PE. When FLR is used, the platform provides the **"ibm,pe-reset-is-flr"** property in the function's OF Device Tree node.
2. The software is responsible for quiescing (stopping) any MMIO *Load* and *Store* activities to the PE prior to resetting the PE.
3. If the platform uses FLR to implement the PE reset, software may need to understand that this is a pulse and not a solid level, such that the adapter is not held at reset during the time from the call with *Function* 1 and the call with *Function* 0.

R1–7.3.11.2–3. For the EEH option: After deactivation of the reset (*Function* 0), software must delay access to that PE until the minimum time after reset that is required for the PE to be come stable has elapsed, as designated by the bus specifications for the particular type bus or buses involved (for example, 1.5 seconds for PCI Express).

Software Implementation Notes:

1. Different implementations of PCI Express may require different amounts of delay in order to traverse the I/O fabric since individual component delays are plug-in card specific.
2. The *ibm,read-slot-reset-state2* RTAS call returns a *PE Reset State* of 5 (PE is unavailable) while any reset delay time is happening for hardware outside the PE.

R1–7.3.11.2–4. For the EEH option: If the *ibm,set-slot-reset* call is called with a *Function* of 0 (deactivate) and any reset to the reset domain specified by the *PE configuration address* is active, then the RTAS call must de-activate all resets to that PE configuration address.

R1–7.3.11.2–5. For the EEH option: If the *ibm,set-slot-reset* call is called with a *Function* of 0 (deactivate) and there is no operation to be performed (for example, the reset to the reset domain specified by the PE configuration address is not active), then the RTAS call must return a *Status* of 0 (success).

R1–7.3.11.2–6. For the EEH option: When the *ibm,set-slot-reset* call is called with a *Function* of 1 or 3 (activate) with a valid PHB Unit ID and *config_addr* and it is the case that FLR is not being used by the platform to reset the PE, then the RTAS call must activate the reset to the reset domain as designated by the *PE configuration address,* if not already activated.

R1–7.3.11.2–7. For the EEH option: When the *ibm,set-slot-reset* RTAS call implements *Function* 3, the platform must also provide the **`"ibm,reset-capabilities"`** property in the **`RTAS`** node of the OF device tree.

R1–7.3.11.2–8. For the EEH option: When the *ibm,set-slot-reset* call is called with a *Function* of 0 (deactivate) with a valid PHB Unit ID and *config_addr* and if the corresponding PE is in the MMIO Stopped or DMA Stopped state, then the RTAS call must bring that PE as designated by the *PE configuration address* out of the MMIO Stopped and DMA Stopped states and clear any applicable platform EEH status state.

R1–7.3.11.2–9. For the EEH option: When the platform uses FLR to reset a PCI Express PE (*ibm,set-slot-reset* call with a *Function* of 1(activate) followed by a call with *Function* 0 (deactivate)), then the platform must provide the **`"ibm,pe-reset-is-flr"`** property in the function's OF Device Tree node, and the platform must always use FLR to reset a PE which contains this property in the OF Device Tree.

R1–7.3.11.2–10. For the EEH option: For a PCI Express PE, the platform must provide the EEH recovery model to the software, as defined by Table 73, "PE State Transition Table," on page 178, regardless of whether Hot Reset or FLR is used to reset the PE.

7.3.11.3 ibm,read-slot-reset-state2

This call queries the state of a PE, and dynamically determines whether a PCI configuration address corresponds to a PE primary bus (that is, if it is the PE configuration address). In addition, when the *PE Reset State* parameter is a 5 (PE is unavailable), then the *PE Unavailable Info* indicates an approximate amount of time for which the PE might be unavailable. The PE configuration address (*PHB_Unit_ID_Hi, PHB_Unit_ID_Low, and config_addr*) for the PE is obtained as defined in Section 7.3.11, "Enhanced I/O Error Handling (EEH) Option Functions," on page 176.

When the *ibm,get-config-addr-info2* RTAS call is implemented, that call can be used instead of this one to determine the PE configuration address. See Section 7.3.11.4, "ibm,get-config-addr-info2," on page 187 and Table 9, "Conventional PCI Express PE Support Summary," on page 72.

R1–7.3.11.3–1. The *ibm,read-slot-reset-state2* call must be implemented using the argument call buffer defined by Table 77, "ibm,read-slot-reset-state2 Argument Call Buffer," on page 185.

Table 77. *ibm,read-slot-reset-state2* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,read-slot-reset-state2* (see Firmware Implementation note, below)
	Number Inputs	3
	Number Outputs	4: Always allowed. 5: May be allowed, depending on the value of the **"ibm,read-slot-reset-state-functions"** property in the **RTAS** node of the device tree (See Section B.6.3.1, "RTAS Node Properties," on page 690).
	Config_addr	Configuration Space Address (Register fields set to 0)
	PHB_Unit_ID_Hi	Represents the most-significant 32-bits of the Unit ID of the PHB that corresponds to the *config_addr*
	PHB_Unit_ID_Low	Represents the least-significant 32-bits of the Unit ID of the PHB that corresponds to the *config_addr*

Table 77. *ibm,read-slot-reset-state2* Argument Call Buffer ***(Continued)***

Parameter Type	Name	Values
Out	*Status*	0: Success -3: Parameter Error
	PE Reset State	Except for a *PE Reset State* of 5, this output is not valid unless the *Config_addr Capabilities* output is a 1 and the *Status* is a 0. 0: Reset deactivated and the PE is not in the MMIO Stopped or DMA Stopped state 1: Reset activated and the PE is not in the MMIO Stopped or DMA Stopped states 2: The PE is in the MMIO Stopped and DMA Stopped states with the reset deactivated and the *Load*/*Store* path is disabled 4: The PE is in the DMA Stopped state with the reset deactivated and the *Load*/*Store* path is enabled 5: PE is unavailable
	Config_addr Capabilities	This output is not valid if the *PE Reset State is a 5.* 0: EEH not supported for the *Config_addr* 1: EEH supported for the *Config_addr*
	PE Unavailable Info	This output is not valid unless the *Config_addr Capabilities* output is a 1 and the *PE Reset State* is a 5 and the *Status* is a 0, in which case the value of this parameter is a 0 if the PE is permanently unavailable, and non-zero if a recovery is in progress and there is an expected availability after the recovery; the non-0 value in this case is the number of milliseconds that the recovery is expected to take.
	PE Recovery Info	This output is only valid if the **"ibm,read-slot-reset-state-functions"** property in the **RTAS** node of the device tree indicates that it is implemented and the call is made with a *Number Outputs* of at least 5 and the *PE Reset State* is a value of 2. This is a 32-bit field with bit significance, as follows: Bits 0:27 - Reserved Bits 28:29 - PE platform reset type. Only valid when bit 31 of this field is a value of 1. Bits 28:29 = 0b00: Firmware does not implement these bits. Reset type is most likely a Hot Reset. Bits 28:29 = 0b01: Platform used a Hot Reset to reset the PE. Bits 28:29 = 0b10: Platform used a Fundamental Reset to reset the PE. Bits 28:29 = 0b11: Platform used an FLR to reset the PE. Bit 30 - Retry Count Hint Bit 30 = 0: Either the PE associated with the *Config_addr* was the source of this PE entering the *PE Reset State* of 2, or the platform has not determined whether this PE was the source or not. Bit 30 = 1: The platform has determined that the PE associated with the *Config_addr* was not the source of this PE entering the *PE Reset State* of 2. That is, setting this bit indicates that this PE entered the *PE Reset State* of 2 as a side-effect of some error outside of this PE's domain. Software may use this hint to not count this occurrence of the *PE Reset State* of 2 as part of any EEH error recovery retry count that it might be keeping for this PE. Bit 31 - Reset Status Bit 31 = 0: PE was not reset as a result of the platform transition to *PE Reset State* of 2. Bit 31 = 1: PE was reset as a result of the platform transition to *PE Reset State* of 2. If the PE is not below a node marked with the special value of the **"status"** property of "reserved", then after deactivation of the platform-initiated PE reset, the platform is required to delay access to that PE until the minimum time after reset that is required for the PE to be come stable has elapsed, as designated by the bus specifications for the particular type bus or buses involved (for example, 1.5 seconds for PCI Express), by returning *PE Reset State* of 5 with *PE Unavailability Info* non-zero (temporarily unavailable) until that time has elapsed). If the PE is below a node marked with the special value of the **"status"** property of "reserved", then after deactivation of the platform-initiated PE reset, the firmware immediately (without delay) transitions the PE to the *PE Reset State* of 2, and it is the controlling software that is required to do the bus-specific delays.

Firmware Implementation Note: The argument call buffer structure and requirements for this call are the same as for the old (removed from this architecture) *ibm,read-slot-reset-state* call, except for the last output parameter. Therefore, it is possible for platforms that still require the old *ibm,read-slot-reset-state* RTAS call to implement

the *ibm,read-slot-reset-state* and *ibm,read-slot-reset-state2* calls with the same RTAS token and use the number of output parameters to determine whether or not to implement the PE *Unavailable Info* parameter.

Platform Implementation Notes:

1. The *ibm,read-slot-reset-state2* RTAS call only returns a *PE Reset State* of 1 (Reset activated and the PE is not in the MMIO Stopped or DMA Stopped state) when the reset may be removed by software; that is, if the error is potentially recoverable. If the firmware has detected a hardware error that is such that the reset to the device cannot be removed or is not safe to remove, the *ibm,read-slot-reset-state2* does not return a *PE Reset State* of 1, but instead returns a *PE Reset State* of 5 (PE is unavailable) along with PE *Unavailable Info* of 0 (PE is permanently unavailable).

2. The *ibm,read-slot-reset-state2* RTAS call should never return a -1 (hardware error), but should instead return a *PE Reset State* of 5 (PE is unavailable) with a *PE Unavailable Info* of 0 (PE is permanently unavailable).

R1–7.3.11.3–2. The *ibm,read-slot-reset-state2* RTAS call must return a *Reset State* value of 5 (PE is unavailable) under any of the following conditions:

a. Firmware has determined that communications with the PE is not available or the path to the PE cannot be traversed at the current time

b. The *ibm,slot-error-detail* RTAS call has been called with a *Function* of 2, and none of the resetting conditions specified in Requirement R1–7.3.11.5–12 have been met.

Software Implementation Notes:

1. The condition under Requirement R1–7.3.11.3–2a may be temporary, with a recovery time in the range of seconds (for example, as little as 3 seconds or up to couple of minutes). Software may chose to delay the time indicated in the *PE Unavailable Info* and issue the *ibm,read-slot-reset-state2* call again when a temporary condition exists. The condition may also be clearable with a power cycle of the PE, in which case the firmware may return a *Status* of 990x to the *set-power-level* RTAS call, to delay long enough to clear the temporary condition.

2. *Config_addr Capabilities* may be indeterminate when the *PE Reset State* of 5 (PE is unavailable) is returned. Software should ignore the *Config_addr Capabilities* return when the *PE Reset State* of 5 is returned.

R1–7.3.11.3–3. If the *ibm,read-slot-reset-state2* RTAS call must return a *PE Reset State* value of 5 (PE is unavailable) then it must indicate in the *PE Unavailable Info* parameter one of the following:

a. A value of zero, if there is no error recovery in progress that makes the PE available in any predictable amount of time (that is, the PE is “permanently” unavailable; for example, until a power cycle or until a repair action).

b. A non-zero value, indicating the approximate time in milliseconds after which the path to the PE is expected to become available again.

R1–7.3.11.3–4. The *ibm,read-slot-reset-state2* RTAS call must return a *Config_addr Capabilities* of 1 (EEH supported for the *Config_addr*) for every *Config_addr* within a PE and for the PE configuration address.

R1–7.3.11.3–5. The *ibm,read-slot-reset-state2* RTAS call must comply with the state transitions defined in Table 73, “PE State Transition Table,” on page 178.

7.3.11.4 ibm,get-config-addr-info2

This call is used obtain information about fabric configuration addresses, given the PCI configuration address. See Section 4.1, “I/O Topologies and Endpoint Partitioning,” on page 71 for more information on PEs and determining PE configuration addresses.

The PCI configuration address (*PHB_Unit_ID_Hi, PHB_Unit_ID_Low, and config_addr*) for the call is defined by Table 34, “Config_addr Definition,” on page 134.

R1–7.3.11.4–1. The *ibm,get-config-addr-info2* call must be implemented using the argument call buffer defined by Table 78, “ibm,get-config-addr-info2 Argument Call Buffer,” on page 188.

Table 78. *ibm,get-config-addr-info2* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,get-config-addr-info2* (see Firmware Implementation note, below)
	Number Inputs	4
	Number Outputs	2
	Config_addr	Configuration Space Address (Register fields set to 0)
	PHB_Unit_ID_Hi	Represents the most-significant 32-bits of the Unit ID of the PHB that corresponds to the *config_addr*
	PHB_Unit_ID_Low	Represents the least-significant 32-bits of the Unit ID of the PHB that corresponds to the *config_addr*
	Function	See Table 79, “ibm,get-config-addr-info2 Function Input and Info Output,” on page 188 for available functions.
Out	*Status*	0: Success -3: Parameter Error
	Info	See Table 79, “ibm,get-config-addr-info2 Function Input and Info Output,” on page 188 for values.

R1–7.3.11.4–2. The *ibm,get-config-addr-info2* RTAS call must return the *Data* output as per Table 79, “ibm,get-config-addr-info2 Function Input and Info Output,” on page 188

Table 79. *ibm,get-config-addr-info2 Function* Input and *Info* Output

Function Input	Definition	*Info* Output
0	Get PE configuration address	PE configuration address (as defined by Table 34, “Config_addr Definition,” on page 134). Result returned in *Info* output is: ◆ Equal to the *Config_addr* input if there is no bridge or switch between the IOA function (endpoint) and the PE primary bus. ◆ Equal to the *Config_addr* of the PE primary bus if there is a bridge or switch between the IOA function and the PE primary bus. ◆ Undefined if *Config_addr* is not in a PE (query for PE state by using *Function* 1 first or by *ibm,read-slot-reset-state2* RTAS call). A *Status* of -3 (Parameter Error) is returned in this case, also.
1	Query shared PE state	0: *Config_addr* is not in a PE (EEH not supported for the *Config_addr*). 1: Not shared (Only one IOA function in the PE). 2: Shared (More than one IOA function in the PE).

7.3.11.5 ibm,slot-error-detail

This call combines device driver information, as gathered by the device driver prior to this call, with information derived by firmware from the platforms I/O infrastructure to create a detailed event log concerning a recoverable EEH event. In this way, both OS and platform maintenance applications have access to all the information about a given event. In addition, the OS can mark a PE configuration address as being in an unavailable state due to excessive errors.

The caller supplies the device driver information, referenced by the *Device_Driver_Error_Buffer* argument. The *Returned_Error_Buffer* argument points to a buffer that this call fills with valid error log data as defined in the error log format section. Different platforms log using different versions of the error logging format. The error log data may include platform specific data as well as device driver data passed in the *Device_Driver_Error_Buffer*. Regardless of the error log version used, the data in the *Returned_Error_Buffer* is in an extended log format as defined in Section 10.3.2, "RTAS Error/Event Return Format," on page 289. When the call returns data for version 6 or greater, the device driver error buffer data is included as the last User Data section. The device driver data in the return buffer may be truncated from what is passed by the device driver or completely eliminated as necessary to ensure that the returned buffer length is not exceeded.

The *Config_addr* supplied is the PE configuration address (*PHB_Unit_ID_Hi, PHB_Unit_ID_Low, and config_addr*) for the PE, obtained as defined in Section 7.3.11, "Enhanced I/O Error Handling (EEH) Option Functions," on page 176, or a configuration address within the PE. The I/O fabric information that is captured by the platform consists of useful PCI configuration state at and above the supplied *Config_addr*.

This RTAS call supports both plug-in PCI cards and built-in PCI IOAs.

In this section, the term unavailable, when applied to a PE, means that *ibm,read-slot-reset-state2* would return a PE Reset State of 5 (PE is unavailable) at the current time.

R1–7.3.11.5–1. For the EEH option: The argument call buffer for the *ibm,slot-error-detail* call must correspond to the definition given in Table 80, "ibm,slot-error-detail Argument Call Buffer," on page 189.

Table 80. *ibm,slot-error-detail* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,slot-error-detail*
	Number Inputs	8
	Number Outputs	1
	Config_addr	Configuration address (Register numbers set to 0)
	PHB_Unit_ID_Hi	Represents the most-significant 32-bits of the Unit ID of the PHB that corresponds to the *config_addr*
	PHB_Unit_ID_Low	Represents the least-significant 32-bits of the Unit ID of the PHB that corresponds to the *config_addr*
	Device_Driver_Error_Buffer	Real address of an error log buffer containing device driver debug data. This data is integrated into the final error log
	Device_Driver_Error_Buffer_Length	Length of the *Device_Driver_Error_Buffer*
	Returned_Error_Buffer	Real address of an error log buffer to contain a compliant error log entry composed by the RTAS
	Returned_Error_Buffer_Length	Length of the *Returned_Error_Buffer*
	Function	1: Temporary Error 2: Permanent Error
Out	*Status*	1: No Error Log Returned 0: Success -1: Hardware Error (cannot create log)

R1–7.3.11.5–2. The *Returned_Error_Buffer* format must be the same as implemented by *event-scan* on the platform.

R1–7.3.11.5–3. To prevent standard error log record truncation, the *Returned_Error_Buffer_Length* must equal the value of the OF device tree property **"rtas-error-log-max"**.

R1–7.3.11.5–4. If the PE corresponding to the *Config_addr* is in the MMIO Stopped or DMA Stopped state, then the *ibm,slot-error-detail* RTAS call must return a *Status* of 0 and an error log that defines the FRU or FRUs to which the error is isolated.

R1–7.3.11.5–5. If the communications with the *Config_addr* is not available, the path to the *Config_addr* cannot be traversed at the current time, or this call has previously made with a *Function* of 2 and none of the conditions that reset this state have been met (that is the PE is unavailable), then the *ibm,slot-error-detail* RTAS call must return a *Status* of 0 and an error log that defines the FRU or FRUs to which the error is isolated.

R1–7.3.11.5–6. If the conditions in Requirements R1–7.3.11.5–4 and R1–7.3.11.5–5 are not met, then the *ibm,slot-error-detail* RTAS call must return a *Status* of 1, no error found, with no error log entry returned.

Software and Platform Implementation Note: In some cases, the platform may return an information-only error log to meet Requirements R1–7.3.11.5–4 and R1–7.3.11.5–5. For example, in some implementations this might be appropriate if the actual error was already logged via another RTAS call or this call was previously made with a *Function* of 2 and none of the conditions that reset this state have been met.

R1–7.3.11.5–7. Once a PE is unavailable and in the absence of any state-resetting action by the OS that clears the corresponding PE configuration address EEH error (for example, reset or power cycle), the platform must return an error log in response to the *ibm,slot-error-detail* RTAS call.

R1–7.3.11.5–8. Once a PE has experienced a state-resetting action by the OS that clears the corresponding PE configuration address EEH error (for example, reset or power cycle), that makes the PE available, the platform must return a *Status* of 1, no error found, with no error log entry in response to the *ibm,slot-error-detail* RTAS call.

R1–7.3.11.5–9. If the *ibm,slot-error-detail* RTAS call *Device_Driver_Error_Buffer_Length* argument is non-zero, indicating the existence of optional device driver error data, the referenced buffer must contain an extended event log as defined in Section 10.3.2, "RTAS Error/Event Return Format," on page 289.

R1–7.3.11.5–10. *(Requirement Number Reserved For Compatibility)*

R1–7.3.11.5–11. When the *ibm,slot-error-detail* RTAS call returns an extended log debug record in the buffer specified by the *Returned_Error_Buffer* argument as mandated by Requirements R1–7.3.11.5–4 and R1–7.3.11.5–5 it must truncate the record at the length specified by the *Returned_Error_Buffer_Length* argument.

R1–7.3.11.5–12. If a *Function* of 2 is passed to the *ibm,slot-error-detail* RTAS call, RTAS must unconditionally set the state of the PE corresponding to the *Config_addr* to permanently unavailable; that is, any subsequent calls to *ibm,read-slot-reset-state2* return a PE Reset State of 5 (PE is unavailable) with the *PE Unavailable Info* argument set to zero.

R1–7.3.11.5–13. RTAS must not change a PE Reset state of permanently unavailable unless one of the following occur:

a. A PCI Hot Plug condition for the slot is encountered (as determined by the power being turned off and then on for the slot)

b. The power domain is power cycled for another reason (for example, a power down of the OS image that owns the IOA)

c. The state is cleared by a partition reboot or a dynamic LPAR reassignment of the PCI configuration address.

R1–7.3.11.5–14. After a PE enters the MMIO and DMA Stopped States due to an error, the platform must keep cached error information relative to that error, for reporting via the *ibm,slot-error-detail* RTAS call, until any one of the following events occurs:

a. The *ibm,slot-error-detail* RTAS call is called and the error information is returned.

b. The reset to the PE is activated via the *ibm,set-slot-reset* RTAS call.

c. The removal of the PE from the DMA Stopped State via *Function* 3 of the *ibm,set-eeh-option* RTAS call.

d. The start of a DR operation as signalled by the calling of *set-indicator* with *isolation-state* set to isolate.

R1–7.3.11.5–15. Prior to calling the *ibm,slot-error-detail* RTAS call, the PE which includes the *Config_addr* must not be in the MMIO Stopped State, if the maximum amount of useful information is to be captured, as defined by Requirement R1–7.3.11.5–16.

R1–7.3.11.5–16. The firmware implementing the *ibm,slot-error-detail* is responsible for gathering the PCI fabric configuration space registers, including those at the specified *Config_addr*, and also any other non-PCI I/O fabric registers that might be useful for debug purposes (for example, internal PHB registers), with the suggested appropriate minimum set of PCI configuration registers captured for each PCI device being as indicated in Table 81, “Suggested Minimum PCI Configuration Registers to Capture for ibm,slot-error-detail,” on page 191.

Table 81. Suggested Minimum PCI Configuration Registers to Capture for *ibm,slot-error-detail*

Data Structure	Offset within the Data Structure	Register
Base PCI Configuration Space Header (for all PCI devices)	0x00	Vendor ID
	0x02	Device ID
	0x04	Command
	0x06	Status
	0x08	Revision ID
	0x09	Class Code
Type 0 Configuration Space Header (for non-PCI bridges only)	0x2C	Subsystem Vendor ID
	0x2E	Subsystem ID
Type 1 Configuration Space Header (for PCI bridges only)	0x1E	Secondary Status
PCI-X Capabilities List (for all PCI-X devices)	0x02	PCI-X Command (Type 0 Configuration Header) PCI-X Secondary Status (Type 1 Configuration Header)
	0x04	PCI-X Status (Type 0 Configuration Header) PCI-X Bridge Status (Type 1 Configuration Header)

Table 81. Suggested Minimum PCI Configuration Registers to Capture for *ibm,slot-error-detail* ***(Continued)***

Data Structure	Offset within the Data Structure	Register
PCI Express Capabilities Structure (for all PCI Express devices)	0x02	PCI Express Capabilities
	0x04	Device Capabilities
	0x08	Device Control
	0x0A	Device Status
	0x0C	Link Capabilities
	0x10	Link Control
	0x12	Link Status
	0x14	Slot Capabilities
	0x18	Slot Control
	0x1A	Slot Status
	0x1C	Root Control
	0x1E	Root Capabilities
	0x20	Root Status
Advanced Error Reporting Capability (for all devices implementing AER)	0x00	PCI Express Enhanced Capability Header
	0x04	Uncorrectable Error Status
	0x08	Uncorrectable Error Mask
	0x0C	Uncorrectable Error Severity
	0x10	Correctable Error Status
	0x14	Correctable Error Mask
	0x18	Advanced Error Capabilities and Control
	0x1C	Header Log
	0x2C	Root Error Command (Root Ports only)
	0x30	Root Error Status (Root Ports only)
	0x34	Correctable Error Source Identification (Root Ports only)
	0x36	Error Source Identification (Root Ports only)

R1–7.3.11.5–17. If the *ibm,slot-error-detail* RTAS call is made with the PE in the PE state of 2 (as defined by Table 73, “PE State Transition Table,” on page 178), then the platform must not remove the PE from that state in order to probe the PCI fabric.

R1–7.3.11.5–18. If the *ibm,slot-error-detail* RTAS call is made with the PE in the PE state of 4 (as defined by Table 73, “PE State Transition Table,” on page 178), then the *ibm,slot-error-detail* RTAS call must return with the PE in the PE state of 4, except that if an error occurs in the course of probing the PCI fabric that requires a reset of the PE by the platform, then discontinue probing, return a Status of 0 or 1 (as appropriate), and return the PE in the PE state of 2.

Software and Platform Implementation Notes:

1. In Requirement R1–7.3.11.5–18, it is possible, as a part of the firmware probing the fabric, that the PE will transition temporarily to a PE state of 2, in the case where another EEH event occurs as part of the firmware probing the fabric. If the EEH event does not require a reset of the PE for these subsequent EEH events, then the firmware may transition the PE back to the PE state of 4, to continue probing. Several of these PE state 4->2->4 events may occur as a result of probing the fabric.

2. In Requirement R1–7.3.11.5–18 if an EEH event occurs as a result of probing that fabric that results in a reset of the PE, the returned PE state of 2 does not necessarily need to be checked for by the software on return from the call. The case where this occurs is expected to be rare, and probably signals a non-transient error. In this case the software can continue on with the recovery phase of the EEH processing, and will eventually hit the same event on further processing.

7.3.12 Bridged-I/O EEH Support Option

The Bridged-I/O EEH Support Option provides RTAS calls for restoring the boot time configuration of EEH error domains that contain multiple IOAs or multi-function IOAs (for example, mult-function I/O cards which are constructed by placing multiple IOAs beneath a PCI to PCI bridge or PCI Express switch). During EEH recovery, the IOA is subject to a full hardware reset. These calls recreate any configuration changes, from full hardware reset, that the firmware normally makes during platform boot prior to turning the IOA over to the client program plus any subsequent changes made via ibm,change-msi. Once these calls restore the IOA initial configuration plus interrupts changes, it is the responsibility of the device driver, as part of its EEH recover procedure, to finish the configuration restoration with any non interrupts changes it makes to the IOA.

Bridge types supported by these calls include PCI to PCI bridges (for example, a PCI to PCI bridge on an I/O plug-in card) and PCI Express bridges and switches.

This option does not address the initialization of bridges and switches which are outside of all PEs. Those are the platform's responsibility.

If there is no supported bridge or switch at the PE configuration address specified by the input parameters, then these calls return a "success" without configuring anything, and therefore these calls can be made for all EEH recovery events, regardless of the type of I/O present. The PE configuration address (*PHB_Unit_ID_Hi, PHB_Unit_ID_Low, and config_addr*) for the PE is obtained as defined in Section 7.3.11, "Enhanced I/O Error Handling (EEH) Option Functions," on page 176

Software Implementation Note: Neither *ibm,configure-bridge* nor *ibm,configure-pe* restores changes to an IOA's post boot configuration registers except as made through the *ibm,change-msi* RTAS call (for example, to the point of being able to issue PCI memory space MMIO operations to the IOA, or perform DMA operations from the IOA). It is the software's responsibility to restore any post boot non interrupt changes it made to the IOA's PCI configuration space registers after calling one of these two RTAS calls.

7.3.12.1 ibm,configure-bridge

R1–7.3.12.1–1. For the Bridged-I/O EEH Support option: The *ibm,configure-bridge* call must implement the argument call buffer defined by Table 82, "ibm,configure-bridge Argument Call Buffer," on page 194.

Table 82. *ibm,configure-bridge* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,configure-bridge*
	Number Inputs	3
	Number Outputs	1
	Config_addr	PE configuration address (Register fields set to 0)
	PHB_Unit_ID_Hi	Represents the most-significant 32-bits of the Unit ID of the PHB that corresponds to the *config_addr*
	PHB_Unit_ID_Low	Represents the least-significant 32-bits of the Unit ID of the PHB that corresponds to the *config_addr*
Out	*Status*	990x: Extended delay where x is a number 0-5 (see Software Implementation note below) 0: Success -3: Parameter Error

Software Implementation Note: When the 990x *Status* is returned, it is suggested that software delay for 10 raised to the x milliseconds (where x is the last digit of the 990x return code), before calling *ibm,configure-bridge* again. However, software may issue the *ibm,configure-bridge* call again either earlier or later than this.

Firmware Implementation Note:

1. This call needs to limit the long busy to 9900-9902 with at most a total of 1/5 second before the *ibm,configure-bridge* succeeds. Any longer delays may cause subsequent hardware or application failures.
2. For hardware errors, return a *Status* of 0 (Success). Hardware errors are subsequently discovered by further accesses to the PE and additional EEH events.

R1–7.3.12.1–2. The caller of *ibm,configure-bridge* must provide the PE configuration address, otherwise the RTAS call returns a -3, “Parameter Error”.

R1–7.3.12.1–3. The *ibm,configure-bridge* call must set up all the PCI to PCI bridges, PCI Express bridges, and PCI Express switches within the PE, the way they were delivered at boot time with any modifications made to it via RTAS calls after boot, and must do so with a single sequence of calls to *ibm,configure-bridge*.

R1–7.3.12.1–4. The *ibm,configure-bridge* call must only return a *Status* of 990x if one of the following conditions is true:

a. The operation was not started.

b. Firmware is able to restart the same call for this PE even when other intervening calls to *ibm,configure-bridge* have occurred (That is, OSs are not required to serialize calls to *ibm,configure-bridge*).

R1–7.3.12.1–5. Software must complete all MMIO operations to the IOAs within a PE prior to calling the *ibm,configure-bridge* RTAS call for a PE and must not issue new MMIO operations to the IOAs within the specified PE until after the RTAS call is complete.

R1–7.3.12.1–6. On return from the *ibm,configure-bridge* RTAS call, the platform must have the PE in the same EEH state (as defined by Table 73, “PE State Transition Table,” on page 178) as when the call was made, except that if an error occurs in the course of probing the PCI fabric that requires a reset of the PE by the platform, then discontinue probing, return a *Status* of 0 or 1 (as appropriate), and return the PE in the PE state of 2.

Software and Platform Implementation Note:

1. Given Requirements R1–7.3.12.1–5 and R1–7.3.12.1–6, it is permissible for the platform to temporarily transition the PE from a PE state of 2 to PE state of 4, if the call is made with a PE state of 2 but the hardware requires a PE state of 4 to get access to the PCI fabric. It is also permissible for the platform to go through several of these state changes during the execution of the call if there are errors that occur during the course of probing the PCI fabric that put the PE back into the PE state of 4.

2. In Requirement R1–7.3.12.1–6 if an EEH event occurs as a result of probing that fabric that results in a reset of the PE, the returned PE state of 2 does not necessarily need to be checked for by the software on return from the call. The case where this occurs is expected to be rare, and probably signals a non-transient error. In this case the software can continue on with the recovery phase of the EEH processing, and will eventually hit the same event on further processing.

7.3.12.2 ibm,configure-pe

This call has about the same semantics as the *ibm,configure-bridge* RTAS call, except that it:

1. Has the additional semantics of bypassing the configuration process if the PE has previously not been reset by the platform as a result of entering the EEH Stopped State.

2. Configures all the configurations spaces within the PE, including those of the endpoint devices within the PE (see Requirement R1–7.3.12.2–3).

Thus, this RTAS call can be made at the beginning of any EEH processing.

R1–7.3.12.2–1. For the Bridged-I/O EEH Support option: The *ibm,configure-pe* call must implement the argument call buffer defined by Table 83, “ibm,configure-pe Argument Call Buffer,” on page 195.

Table 83. *ibm,configure-pe* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,configure-pe*
	Number Inputs	3
	Number Outputs	1
	Config_addr	PE configuration address (Register fields set to 0)
	PHB_Unit_ID_Hi	Represents the most-significant 32-bits of the Unit ID of the PHB that corresponds to the *config_addr*
	PHB_Unit_ID_Low	Represents the least-significant 32-bits of the Unit ID of the PHB that corresponds to the *config_addr*
Out	*Status*	990x: Extended delay where x is a number 0-5 (see Software Implementation note below) 0: Success -3: Parameter Error

Software Implementation Note: When the 990x *Status* is returned, it is suggested that software delay for 10 raised to the x milliseconds (where x is the last digit of the 990x return code), before calling *ibm,configure-pe* again. However, software may issue the *ibm,configure-pe* call again either earlier or later than this.

Firmware Implementation Note:

1. This call needs to limit the long busy to 9900-9902 with at most a total of 1/5 second before the *ibm,configure-pe* succeeds. Any longer delays may cause subsequent hardware or application failures.

2. For hardware errors, return a *Status* of 0 (Success). Hardware errors are subsequently discovered by further accesses to the PE and additional EEH events.

R1–7.3.12.2–2. The caller of *ibm,configure-pe* must provide the PE configuration address, otherwise the RTAS call returns a -3, “Parameter Error”.

R1–7.3.12.2–3. If the specified PE has been configured after the last platform or OS initiated reset to the specified PE with *ibm,configure-connector*, *ibm,configure-bridge*, or *ibm,configure-pe*, then the call must return with a *Status* of 0 (Success) without doing any bridge or switch configuration, otherwise the call must set up the configuration spaces of all the PCI to PCI bridges, PCI Express bridges, PCI Express switches, and endpoint functions within the PE, the way they were delivered at boot time except with all sticky error bits left intact, any changes made by calls to ibm,change-msi retained, and must do so with a single sequence of calls to *ibm,configure-pe*.

Software Implementation Notes:

1. The configuration of endpoint functions (the “and endpoint functions” part) in Requirement R1–7.3.12.2–3 was added to the architecture after the firmware without that functionality in the *ibm,configure-pe* RTAS call was shipping. Therefore, any device driver that might run legacy implementations needs to be prepared to restore all endpoint function config spaces, since the *ibm,configure-pe* RTAS call might not.

2. The *ibm,configure-pe* RTAS call does not restore non-interrupts configuration space changes that were made after boot (that is, under direction of the device driver or OS). Therefore, use of the *ibm,configure-pe* RTAS call does not absolve the device driver or OS from the restoration of non-interrupts the PCI configuration space for changes that were made to the configuration space after boot (see Requirement R1–9.1.8–1).

R1–7.3.12.2–4. The *ibm,configure-pe* call must only return a *Status* of 990x if one of the following conditions is true:

a. The operation was not started.

b. Firmware is able to restart the same call for this PE even when other intervening calls to *ibm,configure-pe* have occurred (That is, OSs are not required to serialize calls to *ibm,configure-pe*).

R1–7.3.12.2–5. Software must complete all MMIO operations to the IOAs within a PE prior to calling the *ibm,configure-pe* RTAS call for a PE and must not issue new MMIO operations to the IOAs within the specified PE until after the RTAS call is complete.

R1–7.3.12.2–6. On return from the *ibm,configure-pe* RTAS call, the platform must have the PE in the same EEH state (as defined by Table 73, “PE State Transition Table,” on page 178) as when the call was made, except that if an error occurs in the course of probing the PCI fabric that requires a reset of the PE by the platform, then discontinue probing, return a *Status* of 0 or 1 (as appropriate), and return the PE in the PE state of 2.

Software and Platform Implementation Note:

1. Given Requirements R1–7.3.12.2–5 and R1–7.3.12.2–6, it is permissible for the platform to temporarily transition the PE from a PE state of 2 to PE state of 4, if the call is made with a PE state of 2 but the hardware requires a PE state of 4 to get access to the PCI fabric. It is also permissible for the platform to go

through several of these state changes during the execution of the call if there are errors that occur during the course of probing the PCI fabric that put the PE back into the PE state of 2.

2. In Requirement R1–7.3.12.2–6 if an EEH event occurs as a result of probing that fabric that results in a reset of the PE, the returned PE state of 2 does not necessarily need to be checked for by the software on return from the call. The case where this occurs is expected to be rare, and probably signals a non-transient error. In this case the software can continue on with the recovery phase of the EEH processing, and will eventually hit the same event on further processing.

7.3.13 Error Injection Option

The Error Injection option (ERRINJCT) allows testing software to check out the OS's error paths. This architecture defines the following abstract error categories:

Fatal: Platform Architectural state has been corrupted to an unknown extent. Further valid processing is not possible.

Recovered Random Event: The Central Electronics Complex (CEC) experienced an anomaly. However, platform architectural state has been preserved/restored. The OS should log the event and continue processing.

Recovered Special Event: The CEC has experienced a statistically significant anomaly. While platform architectural state has been preserved/restored, the OS should log the event and discontinue the use of this processor as soon as possible to avoid a fatal situation.

Corrupted Page: The System Memory page (Up to 4 KB) contains uncorrectable errors. The OS should log the event and avoid accessing this page in the future. The OS recovery is possible given that it can either recover the page from backing storage or isolate the error from unaffected processes.

Corrupted SLB: The processor's Segment Look-aside Buffer is corrupted. The OS should log the event and can recover if it can repopulate the SLB from internal tables.

Translator Failure: The processor's virtual to real translation hardware has failed. The processor's architectural state has been preserved in System Memory. The OS may be able to continue the failed processor's program and log the event on an alternate processor in the future.

IOA Bus Error An error has occurred on the I/O bus on which an I/O Adapter (IOA) is attached. IOA or Device driver recovery from the error is possible if the error is such that it is reported to the IOA. Device driver recovery of the IOA's operations is possible when the error is not reported to the IOA, if the EEH option is implemented and enabled.

The ERRINJCT option RTAS call performs a platform dependent accurate simulation of the abstract error requested. In some cases, the platform hardware actually injects an error into the hardware. In others cases, the platform may simply report the anomaly without generating an error. Additionally, the ERRINJCT option provides access to platform specific error injection logic for the benefit of platform aware test software.

R1–7.3.13–1. For the ERRINJCT option: RTAS must implement the *ibm,open-errinjct* call using the argument buffer defined by Table 84, "ibm,open-errinjct Argument Call Buffer," on page 197.

Table 84. *ibm,open-errinjct* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,open-errinjct*
	Number Inputs	0
	Number Outputs	2

Table 84. *ibm,open-errinjct* Argument Call Buffer ***(Continued)***

Parameter Type	Name	Values
Out	*Open Token*	If *Status* is 0, then use this Open Token for corresponding *ibm,errinjct* and *ibm,close-errinjct* calls
	Status	990x: Extended delay, where x is a number 0-5 (see text) 0: Success -1: Hardware Error -2: Busy, try again later -4: Already open -5: PCI Error Injection is not enabled (not available)

When the 990x *Status* is returned, it is suggested that software delay for 10 raised to the x milliseconds (where x is the last digit of the 990x return code), before calling with the same parameters. However, software may issue the call again either earlier or later than this.

Architecture Note: The output buffer is intentionally reversed from what it should be, according to Requirement R1–7.2.8–1 (that is, *Status* not first output), due to code that was implemented and shipped as defined, above.

R1–7.3.13–2. For the ERRINJCT option: On successful completion of the *ibm,open-errinjct call*, Firmware must return an Open Token which uniquely identifies the caller on following *ibm,close-errinjct* and *ibm,errinjct* calls (Firmware may also need to keep around other information about the caller that uniquely identifies the caller when correlated with the Open Token) and must allocate the ERRINJCT facilities to this caller until this same user calls *ibm,close-errinjct*.

R1–7.3.13–3. For the ERRINJCT option: If the ERRINJCT facility has been previously opened, a call to *ibm,open-errinjct call*, must return a -4.

R1–7.3.13–4. For the ERRINJCT option: RTAS must implement the *ibm,close-errinjct* call using the argument buffer defined by Table 85, “ibm,close-errinjct Argument Call Buffer,” on page 198.

Table 85. *ibm,close-errinjct* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,close-errinjct*
	Number Inputs	1
	Number Outputs	1
	Open Token	Open Token that was returned on the corresponding *ibm,open-errinjct* calls
Out	*Status*	990x: Extended delay where x is a number 0-5 (see text) 0: Success -1: Hardware Error -2: Busy, try again later -4: Close Error (User is not the one that opened the ERRINJCT facility or facility not open)

When the 990x *Status* is returned, it is suggested that software delay for 10 raised to the x milliseconds (where x is the last digit of the 990x return code), before calling with the same parameters. However, software may issue the call again either earlier or later than this.

R1–7.3.13–5. For the ERRINJCT option: If the ERRINJCT facility is not open or was not previously allocated to the user via an *ibm,open-errinjct* call (that is, the Open Token along with any other pertinent data does not

correspond with the user that opened the facility via the *ibm,open-errinjct* call), then a call to *ibm,close-errinjct call*, must return a -4 and the facility must remain open for use by the user that originally opened the facility.

R1–7.3.13–6. For the ERRINJCT option: RTAS must implement the *ibm,errinjct* call using the argument buffer defined by Table 86, “ibm,errinjct Argument Call Buffer,” on page 199.

Table 86. *ibm,errinjct* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,errinjct*
	Number Inputs	3
	Number Outputs	1
	Errinjct Token	Token for the specific error injection class see Requirement R1–7.3.13–8
	Open Token	The Open Token that was returned on the corresponding *ibm,open-errinjct* call
	Working Buffer	Real address of a 1 KB buffer on a 1 KB boundary
Out	*Status*	990x: Extended delay where x is a number 0-5 0: Success -1: Hardware Error -2: Busy, try again later -3 Argument Error (Optional) -4: Call Error (User is not the one that opened the ERRINJCT facility or facility not open)

When the 990x *Status* is returned, it is suggested that software delay for 10 raised to the x milliseconds (where x is the last digit of the 990x return code), before calling with the same parameters. However, software may issue the call again either earlier or later than this.

R1–7.3.13–7. For the ERRINJCT option: If the ERRINJCT facility is not open or was not previously allocated to the user via an *ibm,open-errinjct* call (that is, the Open Token along with any other pertinent data does not correspond with the user that opened the facility via the *ibm,open-errinjct* call), then a call to *ibm,errinjct call*, must return a -4 and the facility must remain open for use by the user that originally opened the facility.

R1–7.3.13–8. For the ERRINJCT option: The platform must include the **`“ibm,errinjct-tokens”`** property as defined below in the **`/rtas`** node (see Section B.6.3.1, “RTAS Node Properties,” on page 690) of the OF device tree with a specification for each implemented error injection class.

R1–7.3.13–9. For the ERRINJCT option: The *errinjct-token-names* must be taken from the list provided in Table 87, “Errinjct-token-names,” on page 199.

Table 87. Errinjct-token-names

Errinjct-token-name	Errinjct function
fatal	Simulate a platform fatal error.
recovered-random-event	Simulate a recovered random event
recovered-special-event	Simulate a recovered special (statistically significant) event

Table 87. Errinjct-token-names

Errinjct-token-name	Errinjct function
corrupted-page	Corrupt the specified location (and potentially surrounding locations up to the containing page)
corrupted-slb	Corrupt the SLB entry associated with a specific effective address.
translator-failure	Simulate a translator failure.
ioa-bus-error	Simulate an error on an IOA bus - 32 bit address specification only.
ioa-bus-error-64	Simulate an error on an IOA bus - 64 bit address specification.
platform-specific	Request the firmware perform a platform specific error injection.
corrupted-dcache-start	Start causing a L1 data cache error
corrupted-dcache-end	Stop causing a L1 data cache error
corrupted-icache-start	Start causing an instruction cache error
corrupted-icache-end	Stop causing an instruction cache error
corrupted-tlb-start	Start corrupting TLB
corrupted-tlb-end	Stop corrupting TLB
upstream-IO-error	Inject I/O error above the IOA

R1–7.3.13–10. For the ERRINJCT option: For the errinjct-tokens implemented RTAS must use the work buffer format specified in Table 88, “Errinjct Work Buffer Formats,” on page 200.

Table 88. Errinjct Work Buffer Formats

Errinjct-token-name	Errinjct work buffer format
fatal	Undefined
recovered-random-event	Undefined
recovered-special-event	“1” for a non-persistent cpu recoverable error “2” for a persistent CPU recoverable error
corrupted-page	The first cell contains the upper 32 bits of the real address to corrupt. The second cell contains the lower 32 bits of the real address to corrupt.
corrupted-slb	The first cell contains the effective address associated with the SLB entry to corrupt
translator-failure	Undefined
ioa-bus-error	The first word contains I/O bus address, word aligned, at which to inject the error. The second word is a mask used to mask off up to 24 of the least significant bits of the address which are not to be used in the comparison of address for error injection (a 0 in a bit position masks off the bit, a 1 in the bit position enables the bit to be used in the compare). The third word is the config_addr on the bus which is to receive the injected error. The fourth word is the *PHB_Unit_ID_Hi* of the PHB that corresponds to the *config_addr*. The fifth word is the *PHB_Unit_ID_Low* of the PHB that corresponds to the *config_addr*. The sixth word defines the specifics of when and what to inject, as follows: See Table 89, “ioa-bus-error Semantics for ioa-bus-error Sixth Word and ioa-bus-error-64 Eighth Word Values 0-19,” on page 202 for values 0 through 19. 20: (Optional) Disable PCI error injection for the specified bus 21: Obtain current error inject values. When RTAS returns SUCCESS in the *Status* field the work buffer field values are populated with the current error injected.

Table 88. Errinjct Work Buffer Formats

Errinjct-token-name	Errinjct work buffer format
ioa-bus-error-64	The first and second words contain the I/O bus address, double word aligned, at which to inject the error. The third and fourth words are a mask used to mask off up to 64 of the least significant bits of the address which are not to be used in the comparison of address for error injection (a 0 in a bit position masks off the bit, a 1 in the bit position enables the bit to be used in the compare). The fifth word is the config_addr of an IOA on the bus which is to receive the injected error. The sixth word is the *PHB_Unit_ID_Hi* of the PHB that corresponds to the *config_addr*. The seventh word is the *PHB_Unit_ID_Low* of the PHB that corresponds to the *config_addr*. The eighth word defines the specifics of when and what to inject, as follows: See Table 89, “ioa-bus-error Semantics for ioa-bus-error Sixth Word and ioa-bus-error-64 Eighth Word Values 0-19,” on page 202 for values 0 through 19. 20: (Optional) Disable PCI error injection for the specified bus 21: Obtain current error inject values. When RTAS returns SUCCESS in the *Status* field the work buffer field values are populated with the current error injected.
platform-specific	See platform firmware documentation (RTAS component specifications) for working buffer format for any particular platform.
corrupted-dcache-start corrupted-dcache-end	The first cell defines the specific action to take: 0: Parity error 1: D-ERAT parity error 2: tag parity error The second cell defines the nature of the error: 0: single 1: solid 2: hang Supported injection modes are hardware specific and all modes may not be supported on all hardware. The first supported injection mode in the above list will be used if an unsupported mode is specified (that is, first single, then solid, then hang-pulse). If none of the above modes are available, then the injection option most similar to single in functionality will be used.
corrupted-icache-start corrupted-icache-end	The first cell defines the specific action to take: 0: parity error 1: I-ERAT parity error 2: cache directory 0 parity error 3: cache directory 1 parity error The second cell defines the nature of the error: 0: single 1: solid 2: hang
corrupted-tlb-start corrupted-tlb-end	The first cell defines the nature of the error: 0: single 1: solid 2: hang Supported injection modes are hardware specific and all modes may not be supported on all hardware. The first supported injection mode in the above list will be used if an unsupported mode is specified (that is, first single, then solid, then hang-pulse). If none of the above modes are available, then the injection option most similar to single in functionality will be used.

Programming Note: Options having a “-start” and corresponding “-end” must be called in pairs on the same processor. The corresponding “-end” option should be called after the injected error has been noticed and processed by the caller. On the same processor, other error inject options should not be called between a “-start” and “-end” sequence. However, it is possible to inject the same type of error multiple times by calling “-start” on that CPU as long as the “nature of error” is “single”. The buffer contents should be the same for a “-start” and corresponding “-end”. While not recommended -end can be replaced with a call to *ibm,close-errinjct*, but improper cleanup the machine may result, with the machine left in an unknown state.

R1–7.3.13–11. For the ERRINJCT option: If the platform notifies the OS of a specific CEC error using the machine check interrupt in response to an *ibm,errinjct* RTAS call, the platform must do so only when the proces-

sor's MSR_{RI} bit is active, unless said error is fatal or involves accessing a storage location that has itself been corrupted or is accessed through a corrupted SLB entry.

R1–7.3.13–12. For the ERRINJCT option with the LPAR option: Hypervisor RTAS must allow a partition to only corrupt its own memory pages.

R1–7.3.13–13. For the ERRINJCT option with the LPAR option: Hypervisor RTAS must allow a partition to inject IOA bus errors only if all of the following are true:

a. The IOA bus is not shared with other partitions.

b. The EEH option is implemented and enabled for the bus on which the error injection is requested.

R1–7.3.13–14. For the ERRINJCT option with the LPAR option: The platform must allow at most one partition to issue platform-specific errinjct calls.

R1–7.3.13–15. For the SPLPAR option: The platform must either implement actual hardware error injection with these interfaces, or must fabricate appropriate partition behavior (machine check, error logs, etc.) as if the hardware error had happened.

R1–7.3.13–16. For the Multi-threading Processor option: All threads on the processor on which the error is injected must be prepared to handle the error.

R1–7.3.13–17. For the Error Injection option: The software using the *ibm,errinjct* call must be prepared to receive a -3 for non-implemented errinjct work buffer formats.

R1–7.3.13–18. For the *ioa-bus-error* and *ioa-bus-error-64* functions of the ERRINJCT option: For each *ibm,errinjct* RTAS call invocation, the platform must inject the error specified in the *working buffer* at most once.

Table 89. *ioa-bus-error* Semantics for *ioa-bus-error* Sixth Word and *ioa-bus-error-64* Eighth Word Values 0-19

Operation	Address Space	Cell Value	Conventional PCI PCI-X Mode 1 PCI-X Mode 2	PCI Express
Load	PCI Memory	0	Inject an Address Parity Error	Inject a TLP ECRC Error[a]
		1	Inject a Data Parity Error	
	PCI I/O	2	Inject an Address Parity Error	
		3	Inject a Data Parity Error	
	PCI Configuration	4	Inject an Address Parity Error	
		5	Inject a Data Parity Error	
Store	PCI Memory	6	Inject an Address Parity Error	Inject a TLP ECRC Error (optional)
		7	Inject a Data Parity Error	
	PCI I/O	8	Inject an Address Parity Error	
		9	Inject a Data Parity Error	
	PCI Configuration	10	Inject an Address Parity Error	
		11	Inject a Data Parity Error	

Table 89. *ioa-bus-error* Semantics for *ioa-bus-error* Sixth Word and *ioa-bus-error-64* Eighth Word Values 0-19 ***(Continued)***

Operation	Address Space	Cell Value	Conventional PCI PCI-X Mode 1 PCI-X Mode 2	PCI Express
DMA read	PCI Memory	12	Inject an Address Parity Error	Inject a TLP ECRC Error
		13	Inject a Data Parity Error	
		14	Inject a Master Abort (no response to IOA) Error	--
		15	Inject a Target Abort	Inject a Completer Abort or Unsupported Request[b]
DMA write	PCI Memory	16	Inject an Address Parity Error	Inject a TLP ECRC Error
		17	Inject a Data Parity Error	
		18	Inject a Master Abort (no response to IOA) Error	--
		19	Inject a Target Abort	Not Applicable

a. For PHB implementations that do not allow injection of a TLP ECRC error into the request, or for the case where the injection would be in violation of Requirement R1–4.4.2.1–5 due to the hardware configuration, the platform should emulate the error by setting the appropriate error state in the PHB when EEH is enabled.
b. Inject the error that is injected on a TCE Page Fault.

Platform Implementation Notes:

- Platforms that implement LPAR normally do not allow any partition to be configured to perform platform-specific errinjct calls since they are capable of crashing the entire complex. However, the should provide special hidden overrides for laboratory testing purposes.

Software and Firmware Implementation Notes:

- When a call to *ibm,errinjct* results in an error injected into a processor, then the error is injected on the same processor as the one that called the *ibm,errinjct* RTAS call, not the processor that called the *ibm,open-errinjct*. The OS could call *ibm,open-errinjct*, *ibm,errinjct*, and *ibm,close-errinjct* from three different processors.
- For usability reasons, the *ibm,close-errinjct* RTAS call should do a reasonable amount of cleanup; turning off error injection where it can. However, since the ERRINJCT option is intended for internal use (that is, not intended to be productized) and since software is allowed to basically set unlimited error injections between the calls to *ibm,open-errinjct* and *ibm,close-errinjct*, the firmware may vary by implementation as to what is cleaned up and what is not. An example of something that might be very difficult to clean up is injection of memory errors. Something that might be easier is to turn off the error injection in all bridges to which the caller has access. Users of the ERRINJCT option should consult the implementation documentation for a particular platform to learn about the level of cleanup that is done in the *ibm,close-errinjct* call for that implementation. In the severe case, a reboot may even be necessary after the *ibm,close-errinjct* in order to clear the error. In other cases it may be possible for the caller to partially disable an error that it has set by setting a benign error (for example, in the PCI error injection case, by setting the error injection for a bus that was previously set to inject an error to an address that will never occur to that IOA).
- Test developers are encouraged not to extensively use the platform-specific option to this function. In general, platform-specific implementation options are not carried forward to new platforms.

7.3.14 Firmware Assisted Non-Maskable Interrupts Option (FWNMI)

The FWNMI option provides firmware support for System Reset interrupts and platform dependent error recovery for recoverable machine checks. The firmware gets control on a non-maskable interrupt (NMI), analyses the condition, and, if the processor was not running inside the hypervisor, reports its findings to the OS. The OS registers system reset and machine check handlers by issuing either the *ibm,nmi-register* or *ibm,nmi-register-2* RTAS call. In addition, with these calls the OS permanently relinquishes to firmware the Machine State Register's Machine Check Enable bit, the two hundred fifty six (256) bytes of the System Reset Interrupt vector starting at real location 0x100, the two hundred fifty six (256) bytes of the Machine Check Interrupt vector starting at real location 0x200, as well as the storage page starting at real location 0x7000. The RTAS firmware records the entry points of the OS notification routines to call to report the results of the firmware's analysis and any attempted recovery should the hardware signal a machine check or system reset interrupt. The results of an error analysis are reported via a standard error log structure as defined in Table 137, "RTAS Event Return Format (Fixed Part)," on page 292. The storage containing the error log structure is subsequently released back to firmware use by the OS after it has completed its event handling by the issuance, from the interrupted processor, of the *ibm,nmi-interlock* RTAS call. Multiple processors of the same OS image may experience fatal events at, or about, the same time. The first processor to enter the machine check handling firmware reports the fatal error. Subsequent processors serialize waiting for the first processor to issue the *ibm,nmi-interlock* call. These subsequent processors report "fatal error previously reported". If, after the firmware makes a Machine Check call back, and before the OS issues the *ibm,nmi-interlock* call, the same processor that is currently holding the storage containing the error log structure receives another Machine Check NMI, the firmware has no choice but to declare the condition fatal, log the result and execute the partition's reboot policy.

When the OS gets control after a machine check, at its registered machine check notification routine, all architected processor registers have been restored to the values they contained when the firmware was notified of the interrupt, except for register R3 which contains a real address that points to a 16 byte structure. The first 8 bytes of this area contains the original contents of R3 and the second 8 bytes contains the fixed portion of the standard error log structure. If firmware is able to immediately make a repair determination, the fixed portion indicates that an additional variable part is present and follows the fixed part per the standard error log structure. For some other errors, the determination of the repair action is delayed, and the firmware reports these determinations asynchronously to handling the machine check. The repair action log is queued in the NVRAM and is reported either in a subsequent *event-scan* if the OS image remains operational, or on a subsequent boot. In no case, does the OS call *check-exception* in its machine check notification routine.

The difference between *ibm,nmi-register* and *ibm,nmi-register-2* is that *ibm,nmi-register* allocates the error reporting structure in RTAS space while *ibm,nmi-register-2* places the error reporting structure in real page 7. New OS designs should use *ibm,nmi-register* since support for *ibm,nmi-register-2* will be terminated at some future date.

As with all first level interrupt service routines, the SPRG-2 register is used to save the state of one general purpose register while the processor computes the location of its state save area.

Implementation Note: An acceptable non-LPAR firmware implementation for the NMI check handlers saves one register in an SPRG-2. Then, using the processor number register, determines an offset into a page 7 table of addresses to the start of a per processor RTAS save area (only need a single register saved per processor), and acquires a lock located in page 7 to serialize the use of the RTAS state save area among potentially competing processors. The MSR_{ME} bit then prevents single processor Machine Check stacking in the interval between the Machine Check call back and the *ibm,nmi-interlock* call. LPAR implementations should minimize potential effects to innocent partitions due to Machine Check Interrupts affecting other partitions.

If the NMI was taken inside the hypervisor, then, if the firmware determines that the condition is recoverable, the hypervisor recovery routine is invoked. If the condition is not recoverable, hypervisor clean up routines establish a safe state and mark the hypervisor return routine to invoke the proper OS registered NMI routine rather than doing the standard hypervisor return.

R1–7.3.14–1. All platforms must implement the FWNMI option.

R1–7.3.14–2. For the FWNMI option: The platform must include the **"ibm,nmi-register"** RTAS function property name in the OF **/rtas** node.

R1–7.3.14–3. For the FWNMI option: The platform must include the **"ibm,nmi-register-2"** RTAS function property name in the OF **/rtas** node if the platform requires support from interim OS versions.

R1–7.3.14–4. For the FWNMI option: RTAS must implement the *ibm,nmi-register* and/or *ibm,nmi-register-2,* calls as appropriate per Requirements R1–7.3.14–2 and R1–7.3.14–3 using the argument buffer defined by Table 90, "ibm,nmi-register or ibm,nmi-register-2 Argument Call Buffer," on page 205.

Table 90. *ibm,nmi-register or ibm,nmi-register-2* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,nmi-register or nmi-register-2*
	Number of Inputs	2
	Number of Outputs	1
	System Reset Notification Routine	Real/Logical address of OS routine to call on a System Reset (in the first 32 MB of memory).
	Machine Check a Notification Routine	Real/Logical address of OS routine to call on a Machine Check (in the first 32 MB of memory).
Out	*Status*	0: Success -1: Hardware Error -3: Parameter Error

R1–7.3.14–5. For the FWNMI option: Once the OS has registered for NMI notification, it must not change the contents of the two hundred fifty six (256) bytes of the NMI interrupt vectors at real locations 0x100 or 0x200 or the memory page starting at real location 0x7000.

R1–7.3.14–6. For the FWNMI option: The Real/Logical address of the registered OS Machine Check and System Reset routines must be in the first 32 MB of the OS's memory address space.

Software Implementation Note: Requirement R1–7.3.14–6 ensures that the registered OS Machine Check and System Reset routines are within the code's RMA.

R1–7.3.14–7. For the FWNMI option: If the OS registered with *ibm,nmi-register*, firmware must not store the state of the processor at the time of interrupt in interrupt vectors at locations 0x100 or 0x200 or the memory page starting at real location 0x7000. Firmware may use RTAS space to store such state data.

R1–7.3.14–8. For the FWNMI option: Once the OS has registered for NMI notification, the platform firmware must intercept all System Reset Interrupts on all of the OS's processors.

R1–7.3.14–9. For the FWNMI option: The platform firmware, for those intercepted System Reset interrupts which platform policy dictate are to be forwarded to the OS, must invoke the OS registered System Reset Interrupt notification point with translate off and all other architected processor registers restored to their state at the time of the System Reset.

R1–7.3.14–10. For the FWNMI option: Once the OS has registered for NMI notification, the platform firmware must intercept all Machine Check Interrupts on all of the OS's processors.

R1–7.3.14–11. or the FWNMI option: The platform must provide a mechanism for the firmware to signal a non-maskable interrupt to each processor in a partition.

R1–7.3.14–12. For the FWNMI option: The platform firmware must analyze all intercepted Machine Check Interrupts, determine if the OS may safely continue using the platform, attempt to recover any corrupted architectural state, and report the results of the recovery attempt to the OS.

R1–7.3.14–13. For the FWNMI option: If the platform firmware, on analyzing an intercepted Machine Check Interrupt, determines that the OS may safely continue using the platform, it must invoke the OS registered Machine Check Interrupt notification point with translate off but all other architected processor registers restored to their state at the time of the Machine Check except that General Purpose Register (GPR) R3 contains the real address of a 16 byte memory buffer containing the original contents of GPR R3 in the first 8 bytes and the RTAS Error Log (fixed part) (per Table 137, “RTAS Event Return Format (Fixed Part),” on page 292) in the second 8 bytes.

R1–7.3.14–14. For the FWNMI option: The maximum time for the platform’s processing of a non-fatal machine check interrupt must be on the order of that taken by the *check-exception* critical call.

R1–7.3.14–15. For the FWNMI option: Once the firmware has reported a “fatal” machine check event to an OS image it must only report “fatal error previously reported” (see Chapter 10, “Error and Event Notification,” on page 281) in response to machine checks on any processor belonging to that image.

R1–7.3.14–16. For the FWNMI option: If the platform firmware, on analyzing an intercepted Machine Check Interrupt, determines that the OS may not safely continue using the processor (for example a check stop will certainly result), it must select one of the implementation options given in Table 91, “Unsafe Processor Recovery Options,” on page 206.

Table 91. Unsafe Processor Recovery Options

Option Number	Implementation Option for handling an unsafe processor.
1	Invoke the registered Machine Check Interrupt notification point on a spare processor which platform firmware substitutes for the offending processor. **Note:** Firmware must adjust all interrupt XIVT entries and APM registers, etc., so that the OS need not be aware of the processor substitution. The VPD of the new and old processors are different, the dynamic VPD collection RTAS call can be used to determine the new values. Since the results of this substitution are indicated as a non-fatal error to the OS, the substitution may take no more than 10 times the length of time of a critical check exception process. The firmware makes a best effort to load the decrementer with a value that represents the value in the failed processor at the time of the machine check minus a value that represents the time taken by the substitution process.
2	Mark the processor unsafe, do not return to the OS on that processor and notify the OS to at the next event scan time with a fatal return message. **Note:** This action may cause the OS to “hang” due to locks held by the failing processor etc. that may cause a surveillance time out. The NVRAM firmware error log retains a trail of this condition for reading and logging at the subsequent OS boot. However, in those cases where a hang does not happen, the OS can select some other processor to pick up the thread of execution.

R1–7.3.14–17. For the FWNMI option: RTAS must implement the *ibm,nmi-interlock* call using the Argument buffer defined in Table 92, “ibm,nmi-interlock Argument Call Buffer,” on page 207 which causes the release of the machine check work and reporting area in page 7.

Table 92. *ibm,nmi-interlock* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,nmi-interlock*
	Number of Inputs	0
	Number of Outputs	1
Out	*Status*	0: Success -1: Hardware Error

R1–7.3.14–18. For the FWNMI option: The *ibm,nmi-interlock* RTAS call must not require serialization with respect to any other RTAS or hypervisor calls.

R1–7.3.14–19. For the FWNMI option: The processor receiving the nmi signal must, after it has processed the buffer pointed to by its R3 register, call the *ibm,nmi-interlock* RTAS call.

7.3.15 Memory Statistics

Depending upon the platform configuration, various portions of installed platform memory are in one of several states. Some memory may be mapped out of the address space due to an error in one or more locations. Other memory is used by the platform firmware. What is left is allocated to one or more logical partitions or held in reserve. The usage of memory is a first order platform management parameter, and is needed by platform managers. However, it may also become a covert channel between logical partitions. Therefore, the memory usage information that is surfaced to an OS image by firmware is restricted to total platform memory installed, plus three sub-divisions which total to the total memory installed. These three sub-divisions are the total memory the platform mapped out due to hardware failure, total memory reserved for platform firmware and other partitions, and the memory allocated to the calling OS image. LPAR machines can provide a more detailed memory usage report via their Hardware Management Console. The total memory allocated to the calling OS image is obtained through the device tree (potentially modified by post boot dynamic reconfiguration).

7.3.16 System Parameters Option

The system parameters which are defined are shown in Table 93, “Defined Parameters,” on page 207.

Table 93. Defined Parameters

Parameter token	Parameter	Description	Values	Notes
0	HMC 0			
1	HMC 1			
2 thru 15	HMC 2 thru 15			
16	Reserved			
17	Reserved			
18	Processor CoD Capacity Card Info	See Section 7.3.16.4.1, “CoD Capacity Card Info,” on page 213		1

Table 93. Defined Parameters

Parameter token	Parameter	Description	Values	Notes
19	Memory CoD Capacity Card Info	See Section 7.3.16.4.1, “CoD Capacity Card Info,” on page 213		1
20	SPLPAR Characteristics	Opaque ASCII NULL terminated string		1, 2
21	partition_auto_restart	See Section 7.3.16.5, “Restart Parameters,” on page 221		
22	platform_auto_power_restart	See Section 7.3.16.5, “Restart Parameters,” on page 221		
23	sp-remote-pon	Remote Power On (see Section 7.3.16.6, “Remote Serial Port System Management Parameters,” on page 222)	One byte decimal 0 (for off) 1 (for on) Default 0	
24	sp-rb4-pon	Number of rings until power on (see Section 7.3.16.6, “Remote Serial Port System Management Parameters,” on page 222)	One byte decimal	
25	sp-snoop-str	Snoop sequence string (see Section 7.3.16.7, “Surveillance Parameters,” on page 222)		
26	sp-serial-snoop	Serial snoop enable/disable (see Section 7.3.16.7, “Surveillance Parameters,” on page 222)	0 (for off) 1 (for on) Default 0	
27	sp-sen	Surveillance enable/disable (see Section 7.3.16.7, “Surveillance Parameters,” on page 222)	0 (for off) 1 (for 0n) Default 0	
28	sp-sti	Surveillance time interval in minutes (see Section 7.3.16.7, “Surveillance Parameters,” on page 222)	1-255 Default 5	
29	sp-sdel	Surveillance delay in minutes (see Section 7.3.16.7, “Surveillance Parameters,” on page 222)	1-120 Default 10	
30	sp-call-home	See Section 7.3.16.8, “Call Home Parameter,” on page 222		
31	sp-current-flash-image	See Section 7.3.16.9, “Current Flash Image Parameter,” on page 224	0 (for perm) 1 (for temp)	
32	platform-dump-max-size	See Section 7.3.16.10, “Platform Dump Max Size Parameter,” on page 224	64-bit integer	The value consists of a 32-bit high value followed by a 32-bit low value. The resulting 64-bit value is unsigned.
33	epow3-quiesce-time	See Section 7.3.16.11, “Storage Preservation Option System Parameters,” on page 225	0-65535 seconds Default 0	
34	memory-preservation-boot-time	See Section 7.3.16.11, “Storage Preservation Option System Parameters,” on page 225	0-65535 seconds Default 0	

Table 93. Defined Parameters

Parameter token	Parameter	Description	Values	Notes
35	SCSI Initiator Identifier	See Section 7.3.16.12, “SCSI Initiator Identifier System Parameters,” on page 225		
36	AIX support			
37	Enhanced Processor CoD Capacity information	See Section 7.3.16.4.3, “Enhanced CoD Capacity Info,” on page 214		1
38	Enhanced Memory CoD Capacity information			1
39	CoD options	See Section 7.3.16.13, “CoD Options,” on page 228.		1
40	Platform Error Classification	See Section 7.3.16.14, “Platform Error Classification,” on page 228.		
41	Firmware Boot Options	See Section 7.3.16.15, “Firmware Boot Options,” on page 228		
42	platform-processor-diagnostics-run-mode	See Section 7.3.16.16, “Platform Processor Diagnostics Options,” on page 229	One byte decimal 0=disabled 1=staggered 2=immediate 3=periodic	
43	Processor Module Information	See Section 7.3.16.17, “Processor Module Information,” on page 230		1
44	Cooperative Memory Over-commitment Definitions	Opaque ASCII NULL terminated string		1, 3
45	Cede Latency Settings Information	See Section 7.3.16.18, “Cede Latency Settings Information,” on page 230		
46	Target Active Memory Compression Factor	See Section 7.3.16.19, “Target Active Memory Compression Factor,” on page 231	Field length: 2 bytes Format: binary Range: 100 -- 1000	
47	Performance boost modes vector	See Section 7.3.16.20, “Performance Boost Modes Vector,” on page 231	From *ibm,get-system-parameter* the field length is 96 bytes consisting of 3 32 byte bit vectors. To *ibm,set-system-parameter* the field is a single 32 byte bit vector	
48	UUID	16 Byte String	See Section 7.3.16.21, “Universally Unique IDentifier,” on page 233	
>48	Reserved			

Notes:

1. These system parameters are defined for the *ibm,get-system-parameter* RTAS call only. An attempt to set them using the *ibm,set-system-parameter* RTAS call results in a return *Status* of -9002 (Setting not allowed/authorized).

2. The format of the SPLPAR string is beyond the scope of this architecture. See also, Appendix A, “SPLPAR Characteristics Definitions,” on page 657.

3. See Appendix I, “CMO Characteristics Definitions,” on page 839.

Further parameters will be defined as required.

R1–7.3.16–1. All platforms must support the System Parameters option.

R1–7.3.16–2. *(Requirement Number Reserved For Compatibility)*

R1–7.3.16–3. For the System Parameters option: If the length of the data for a parameter in Table 93, “Defined Parameters,” on page 207 is less than what is specified in the requirements for a parameter or if the data value in an *ibm,set-system-parameter* RTAS call is other than what is allowed by the requirements for the parameter, the platform must return a -9999 indicating a parameter error.

R1–7.3.16–4. For the System Parameters option: The default values defined for parameters sp-sen, sp-sti and sp-sdel in the Table 93, “Defined Parameters,” on page 207 must apply to the platform prior to any *ibm,set-system-parameter* RTAS call.

R1–7.3.16–5. For the System Parameters option: The *ibm,get-system-parameter* RTAS call must implement the argument call buffer defined by Table 94, “ibm,get-system-parameter Argument Call Buffer,” on page 211. If the *ibm,set-system-parameter* RTAS call is implemented, it must use the argument call buffer defined by Table 95, “ibm,set-system-parameter Argument Call Buffer,” on page 212.

R1–7.3.16–6. For the System Parameters option: If the platform implements the *ibm,set-system-parameter* RTAS call it must also implement the *ibm,get-system-parameter* RTAS call.

R1–7.3.16–7. For the System Parameters option: A system parameter, which is not supported by the system, must return a *Status* of -3 (System parameter not supported) from the RTAS call.

R1–7.3.16–8. For the System Parameters option: A system parameter for which access is not authorized, must return a *Status* of -9002 (Not authorized) from the RTAS call.

R1–7.3.16–9. For the System Parameters option: When a platform implements a system parameter, it must meet the definition in Table 93, “Defined Parameters,” on page 207 including applicable descriptions and notes.

R1–7.3.16–10. For the System Parameters option: An *ibm,get-system-parameter* RTAS call with a buffer length of zero (0) must return a *Status* of 0 (success) if the parameter is supported and authorized, a *Status* of -3 if not supported, or a *Status of* -9002 if not authorized.

R1–7.3.16–11. For the System Parameters option: An *ibm,set-system-parameter* RTAS call with a parameter length of zero (0) must return a *Status* of 0 (success) if the parameter is supported and authorized, a *Status* of -3 if not supported, or a *Status* of -9002 if not authorized.

Programming Note: A partition may lose or gain authority for an *ibm,get-system-parameter* or *ibm,set-system-parameter* call dynamically. For instance, three consecutive calls with the same parameters could return *Status* of success, not authorized, and success.

R1–7.3.16–12. For the System Parameters option: The platform must enforce the length of system parameter strings as follows: input strings to ibm,set-system-parameters not to exceed 1024 bytes in length else the platform returns a *Status* of -9999 (parameter error) from the RTAS call; output strings from ibm,get-system-parameters not to exceed 4000 bytes.

R1–7.3.16–13. For the System Parameter option with the SPLPAR option: The Platform must implement parameter token 20 as defined in Table 93, “Defined Parameters,” on page 207 for *ibm,get-system-parameter*.

Implementation Note: Of course the OS is allowed to provide and specify a buffer that is larger than the maximum system parameter length.

7.3.16.1 ibm,get-system-parameter

Table 94. *ibm,get-system-parameter* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,get-system-parameter*
	Number Inputs	3
	Number Outputs	1
	Parameter	Token of system parameter to retrieve
	buffer	Real address of data buffer
	length	length of data buffer
Out	*Status*	0: Success -1: Hardware Error -2: Busy, Try again later -3: System parameter not supported -9002: Not authorized -9999: Parameter Error 990x: Extended delay where x is a number 0-5 (see text below)

The *ibm,get-system-parameter* RTAS call fetches the data for the selected parameter and places it at the address specified in the buffer operand. The first two (2) bytes of the data in the buffer are the length of the returned data, not counting these first two (2) bytes. The length of string data includes the length of the NULL but excludes the length field. If the buffer length is less than the returned data length, the data is truncated at the end of the buffer. The maximum length of the input parameter data string for *ibm,set-system-parameter* is architecturally limited to 1024 bytes of data and 2 bytes of length, totaling 1026 bytes. The maximum length of the output parameter data string for ibm,get-system-parameter is architecturally limited to 4000 bytes of data and 2 bytes of length, totaling 4002 bytes. The only currently valid parameters are as specified in Table 93, “Defined Parameters,” on page 207.

When the 990x *Status* is returned, it is suggested that software delay for 10 raised to the x milliseconds (where x is the last digit of the 990x return code), before calling *ibm,get-system-parameter* with the same parameter index. However, software may issue the *ibm,get-system-parameter* call again either earlier or later than this.

7.3.16.2 ibm,set-system-parameter

Table 95. *ibm,set-system-parameter* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,set-system-parameter*
	Number Inputs	2
	Number Outputs	1
	Parameter	Token number of the target system parameter
	buffer	Real address of data buffer
Out	*Status*	0: Success -1: Hardware Error -2: Busy, Try again later -3: System parameter not supported -9002: Setting not allowed/authorized -9999: Parameter Error 990x:Extended delay where x is a number 0-5 (see text below)

The *ibm,set-system-parameter* RTAS call fetches the data from the address specified in the buffer operand and sets it into the system parameter specified by the *Parameter* operand. The first two (2) bytes of the data in the buffer are the length of the data, not counting these first two (2) bytes. The length of string data includes the length of the NULL but excludes the length field. The only currently valid parameters are as specified in Table 93, “Defined Parameters,” on page 207.

When the 990x *Status* is returned, it is suggested that software delay for 10 raised to the x milliseconds (where x is the last digit of the 990x return code), before calling *ibm,set-system-parameter* with the same parameter index. However, software may issue the *ibm,set-system-parameter* call again either earlier or later than this.

7.3.16.3 HMC Parameter

The full HMC parameter data string is returned when the *ibm,get-system-parameter* RTAS call is issued.

HMC parameter contents are:
length - two byte binary length of data string associated with the HMC.
System Parameter Data - string of semicolon delimited ASCII data.

The *ibm,set-system-parameter* is not required to support the HMC parameter since the HMC parameter data is set only by the HMC through an out-of-band path. The *ibm,get-system-parameter* RTAS call is provided for reading the parameter data. The RTAS call is available in both LPAR mode and non-LPAR mode.

R1–7.3.16.3–1. For the HMC Parameter: If the *ibm,set-system-parameter* RTAS call is provided, the use of the HMC parameters, 0-15, must always return not authorized *Status*, -9002.

R1–7.3.16.3–2. For the HMC Parameter: The format of each HMC system parameter supported by this system must consist of a two byte binary length field describing the full length of the parameter data, followed by a series of variables where each variable is of the form “keyword” followed by “=” followed by “value” and terminated by “;”. The order of the variables is undefined and the total number of variables is undefined. The platform must provide the “HscIPAddr” keyword[1], and the “RMCKey” keyword. The data after the equal sign (=) may or may not have content. The value of the “HscIPAddr” keyword is the IP address of that HMC. The value of the “RMCKey” keyword is either null or is the RMC key for that system or partition. The value

of a keyword is null if there is nothing between the "=" and the ";". The update state of keyword values of an HMC system parameter are uncertain when the HMC stops talking to the managed system.

R1–7.3.16.3–3. For the HMC Parameter: All HMC system parameter data must be printable ASCII characters, excluding the two byte binary length field.

R1–7.3.16.3–4. For the HMC Parameter: The lowest valued HMC system parameter which returns a -3 *Status* must have no higher valued HMC system parameter which is supported. That is, a scan of HMC system parameters from 0 until the first -3 *Status* must indicate all supported HMC system parameters.

R1–7.3.16.3–5. For the HMC Parameter: If there is no HMC control of this platform, the platform must return a null response, zero length data, to requests for all supported HMC system parameters.

Implementation Note: Since the system is not necessarily to be HMC controlled, it is shipped with the HMC parameter set to the zero (0) length. If the system is HMC controlled, the HMC passes the parameter values to the system at boot time so that the *ibm,get-system-parameters* RTAS call indicates HMC control. If there is deconfigured the HMC can write the zero (0) length data to the system. If that is not done, the system can write the parameter to zero length on a hard reset and the HMC, if present, then initializes the data.

R1–7.3.16.3–6. For the HMC Parameter: The platform must truncate the HMC system parameter data at the buffer length if the buffer length is less than the data length plus 2.

7.3.16.4 Capacity on Demand (CoD) Option

Platforms may optionally provide mechanisms for securely licensing a subset of the platform's physically installed resources for use. The CoD option includes system parameters relating to the CoD Capacity "smart card" which is used to securely store and validate the license information. Dynamically adding memory and cpu resources to running partitions requires the CoD option combined with the Logical Resource Dynamic Reconfiguration option.

Additionally platforms may provide a provisional CoD activation mode known as "Trial CoD". This mode provides immediate availability of resources while the permanent license is on order. The CoD resources are made available for a platform dependent period of power on hours.

If the platform implements the CoD sparing option and the platform predicts the failure of a CoD resource, given that there is spare capacity of that resource, the platform makes available a spare resource so that the OS can migrate work off the failing resource and return the failing resource to the OS. If the OS takes advantage of this sparing, by actually using the available resource the OS is using resources in excess of the permanently licensed entitlement until the failing CoD resource is returned to the platform.

R1–7.3.16.4–1. For the CoD option: The platform must support the System Parameter option (*ibm,get-system-parameter*) along with Parameter tokens 18 and 19 as described in Table 93, "Defined Parameters," on page 207.

7.3.16.4.1 CoD Capacity Card Info

Software Note: System parameters 18 and 19 present only permanently activated capacity. These parameters will be removed at some point in the future. OSs should begin using the enhanced CoD Capacity parameters.

These two read only system parameters (one for memory and another for processors) are ASCII hexadecimal strings representing the current licensed entitlement of CoD resources of their respective types. These strings contains 9 packed fields as presented in Table 96, "CoD Capacity Card Info String Packed Fields," on page 214.

1.The acronym "HSC" was replaced by "HMC" but this keyword was retained with "Hsc" so as to not invalidate code already created using the keyword.

Table 96. CoD Capacity Card Info String Packed Fields

Field Number	Definition
1	System type (4 ASCII characters) (4 bytes)
2	System serial number (8 ASCII characters: pp-sssss) (8 bytes)
3	CoD capacity card Custom Card Identification Number (CCIN) (4 ASCII characters) (4 bytes)
4	CoD capacity card serial number (10 ASCII characters: pp-sssssss) (10 bytes)
5	CoD capacity card unique ID (16 ASCII characters) (16 bytes)
6	CoD resource identifier (4 ASCII characters) (4 bytes)
7	Quantity of Activated CoD resource (4 characters ASCII) (4 bytes)
8	CoD sequence number (4 numeric ASCII characters) (4 bytes)
9	CoD activation code entry check (1 byte hex check sum, two ASCII characters - based on EBCDIC representation of items: 1, 2, 3, 4, 5, 6, 7, and 8) (2 bytes)

R1–7.3.16.4.1–1. For the CoD option: The platform's *ibm-get-system-parameter* RTAS call, specifying the CoD Capacity Card Info, must, upon successful completion, return the ASCII representation of the information defined in Table 96, "CoD Capacity Card Info String Packed Fields," on page 214 for the managed CoD resource type specified by the system parameter token.

R1–7.3.16.4.1–2. For the CoD option: The platform's *ibm,set-system-parameter* RTAS call specifying the CoD Capacity Card Info, must not return a *Status* of 0 (success); the expected return is a *Status* of -9002 (Setting not allowed/authorized), however, under special cases a *Status* of -1 (Hardware error), or one of the Busy or Extended Delay return *Status* return values is allowed.

7.3.16.4.2 Predictive Failure Sparing with Free Resources

A platform may optionally provide an unused resource to a partition that is notified of a predictive failure. This allows the partition's OS to transparently substitute the spare resource for the failing one in some situations. To take advantage of this situation, the partition's OS queries the free DR slot(s) of the resource type to determine if a spare resource is available, and if so uses the other DR RTAS calls to acquire the resource. In some cases resources are free because they have not been assigned to partitions.

A platform may optionally provide an unused CoD resource to a partition as a predictive failure spare. In such cases, the result of an *get-sensor-state* (entity-sense) for the DR slot returns the state of "exchange". Between the time that the OS takes ownership, via *set-indicator* (allocation-state, exchange), of the spare CoD resource available and the OS gives up the failing resource, the platform exceeds the licensed entitlement for that resource.

R1–7.3.16.4.2–1. For the Predictive Failure Sparing option: The platform, upon provisionally making available a spare CoD resource in response to a predictive failure, must set the CoD Resource Provisional Activation timer, to time out the use of the provisionally activated excess resources.

7.3.16.4.3 Enhanced CoD Capacity Info

These two read only system parameters (one for processor and one for memory) return ASCII hexadecimal strings representing the current licensed CoD resources.

The strings are constructed with a fixed "base" section followed by zero or more optional sections. The definitions below show all optional sections. The caller should not expect the presence or order of all optional sections. Each optional section starts with the following 3 members:

- Offset to next section (zero for last section)
- Size in bytes of the section (including these three members)
- Name of the section

The specific meaning of the members of each section is beyond the scope of this architecture; refer to the specific platform design documents.

All data in these tables are composed of printable ASCII characters. There are no NULLs or other non-printable characters.

Programming Note: On a platform where the base processor capacity is a fraction of a full processor, the data in the BaseProc section below is rounded up to the next larger whole number.

R1–7.3.16.4.3–1. For the CoD option: The platform's *ibm,get-system-parameter* RTAS call, specifying the Enhanced CoD Processor Capacity Info, must, upon successful completion, return the ASCII representation of the information defined in Table 97, “Enhanced CoD Processor Capacity Info, Version 1,” on page 215 for managed CoD processor resources.

Table 97. Enhanced CoD Processor Capacity Info, Version 1

Offset	Size in Bytes	Section	Description	Format
0	4	Meta	Table version indicator	4 ASCII characters “V1 “
4	4	Meta	Decimal number of optional sections	4 ASCII numeric characters
8	4	Base	Decimal offset in bytes from the start of this section to the start of the next section (that is, the offset from the first byte of this member to the first byte of the next section). Zero if the last section	4 ASCII numeric characters
12	4	Base	Section Length -- The length of this section in bytes (including offset member above)	4 ASCII numeric characters “78 “
16	8	Base	Section Name	8 ASCII characters “BASE “
24	4	Base	system type	4 ASCII characters
28	10	Base	System serial number	10 ASCII characters: “pp-ssssss “
38	4	Base	CoD capacity card CCIN	4 ASCII characters
42	10	Base	CoD capacity card serial number	10 ASCII characters “hh-hssssss”
52	16	Base	CoD capacity card unique ID	16 ASCII characters
68	4	Base	CoD resource identifier	4 ASCII characters
72	4	Base	Quantity of permanently activated resources	4 ASCII characters
76	4	Base	CoD sequence number	4 numeric ASCII characters
80	2	Base	CoD activation code entry check	1 byte hex check sum, 2 ASCII characters -- based on EBCDIC representation of previous 8 entries.
82	4	Base	Total CoD resources installed in system	4 numeric ASCII characters

Table 97. Enhanced CoD Processor Capacity Info, Version 1 ***(Continued)***

Offset	Size in Bytes	Section	Description	Format
On/Off Processor Resources				
0	4	OnOffPrc	Decimal offset in bytes from the start of this section to the start of the next section (that is, the offset from the first byte of this member to the first byte of the next section). Zero if the last section.	4 ASCII numeric characters
4	4	OnOffPrc	Section Length -- The length of this section in bytes (including offset member above)	4 ASCII numeric characters “66 “
8	8	OnOffPrc	Section Name	8 ASCII characters “ONOFFPRC”
16	1	OnOffPrc	On/Off CoD enabled	1 ASCII character '0' or '1'
17	1	OnOffPrc	On/Off CoD active	1 ASCII character '0' or '1'
18	4	OnOffPrc	On/Off CoD feature	4 ASCII characters
22	4	OnOffPrc	On/Off CoD activated resources	4 ASCII numeric characters
26	4	OnOffPrc	On/Off CoD sequence number	4 ASCII numeric characters
30	2	OnOffPrc	On/Off CoD checksum	2 ASCII characters
32	4	OnOffPrc	On/Off CoD resources requested	4 ASCII numeric characters
36	4	OnOffPrc	On/Off CoD days requested	4 ASCII numeric characters
40	4	OnOffPrc	On/Off CoD resource days expired	4 ASCII numeric characters
44	4	OnOffPrc	On/Off CoD resource days remaining	4 ASCII numeric characters
48	4	OnOffPrc	On/Off CoD counter	4 ASCII numeric characters
52	4	OnOffPrc	On/Off standby resources available	4 ASCII numeric characters
56	1	OnOffPrc	On/Off reserved byte	1 ASCII blank
57	4	OnOffPrc	On/Off history of requested resource days	4 ASCII characters
61	1	OnOffPrc	On/Off reserved byte	1 ASCII blank
62	4	OnOffPrc	On/Off history of unreturned resource days	4 ASCII characters
Debit Processor Resources				
0	4	DebitPrc	Decimal offset in bytes from the start of this section to the start of the next section (that is, the offset from the first byte of this member to the first byte of the next section). Zero if the last section.	4 ASCII numeric characters
4	4	DebitPrc	Section Length -- The length of this section in bytes (including offset member above)	4 ASCII numeric characters “82 “
8	8	DebitPrc	Section Name	8 ASCII characters “DEBITPRC”
16	1	DebitPrc	Debit CoD enabled	1 ASCII character '0' or '1'
17	1	DebitPrc	Debit CoD active	1 ASCII character '0' or '1'
18	4	DebitPrc	Debit CoD feature	4 ASCII characters

Table 97. Enhanced CoD Processor Capacity Info, Version 1 ***(Continued)***

Offset	Size in Bytes	Section	Description	Format
22	4	DebitPrc	Debit CoD activated resources	4 ASCII numeric characters
26	4	DebitPrc	Debit CoD sequence number	4 ASCII numeric characters
30	2	DebitPrc	Debit CoD checksum	2 ASCII characters
32	4	DebitPrc	Debit CoD resources requested	4 ASCII numeric characters
36	12	DebitPrc	Debit Reserved	12 ASCII blanks
48	4	DebitPrc	Debit counter	4 ASCII characters
52	4	DebitPrc	Debit standby resources available	4 ASCII numeric characters
56	1	DebitPrc	Debit reserved byte	1 ASCII blank
57	4	DebitPrc	Debit history of expired resource days	4 ASCII numeric characters
61	1	DebitPrc	Debit reserved byte	1 ASCII blank
62	4	DebitPrc	Debit history of unreturned resource days	4 ASCII characters
66	8	DebitPrc	Extended total history of requested On/Off Processor days	8 ASCII characters
74	8	DebitPrc	Extended total history of unreturned On/Off Processor days	8 ASCII characters
			Trial Processor Resources	
0	4	TrialPrc	Decimal offset in bytes from the start of this section to the start of the next section (that is, the offset from the first byte of this member to the first byte of the next section). Zero if the last section.	4 ASCII numeric characters
4	4	TrialPrc	Section Length -- The length of this section in bytes (including offset member above)	4 ASCII numeric characters “66 “
8	8	TrialPrc	Section Name	8 ASCII characters “TRIALPRC”
16	1	TrialPrc	Trial CoD enabled	1 ASCII character '0' or '1'
17	1	TrialPrc	Trial reserved	1 ASCII blank
18	4	TrialPrc	Trial CoD feature	4 ASCII characters
22	4	TrialPrc	Trial CoD activated resources	4 ASCII numeric characters
26	4	TrialPrc	Trial CoD sequence number	4 ASCII numeric characters
30	2	TrialPrc	Trial CoD checksum	2 ASCII characters
32	8	TrialPrc	Trial reserved bytes	8 ASCII blanks
40	4	TrialPrc	Trial days expired	4 ASCII numeric characters
44	4	TrialPrc	Trial days remaining	4 ASCII numeric characters
48	14	TrialPrc	Trial reserved bytes	14 ASCII blanks
62	4	TrialPrc	Trial unreturned resources	4 ASCII numeric characters

Table 97. Enhanced CoD Processor Capacity Info, Version 1 ***(Continued)***

Offset	Size in Bytes	Section	Description	Format
Base Processor Resources				
0	4	BaseProc	Decimal offset in bytes from the start of this section to the start of the next section (that is, the offset from the first byte of this member to the first byte of the next section). Zero if the last section.	4 ASCII numeric characters
4	4	BaseProc	Section Length -- The length of this section in bytes (including offset member above)	4 ASCII numeric characters “20 “
8	8	BaseProc	Section Name	8 ASCII characters “BASEPROC”
16	4	BaseProc	Number of Non-CoD “Base” processors on this system. **NOTE:** If this section is not present, there are no “base” processors and all processors are CoD activated.	4 ASCII numeric characters

R1–7.3.16.4.3–2. For the CoD option: The platform's *ibm,get-system-parameter* RTAS call, specifying the Enhanced CoD Memory Capacity Info, must, upon successful completion, return the ASCII representation of the information defined in Table 98, “Enhanced CoD Memory Capacity Info, Version 1,” on page 218 for the managed CoD memory resources.

Table 98. Enhanced CoD Memory Capacity Info, Version 1

Offset	Size in Bytes	Section	Description	Format
0	4	Meta	Table version indicator	4 ASCII characters “V1 ”
4	4	Meta	Decimal number of optional sections	4 ASCII numeric characters
8	4	Base	Decimal offset in bytes from the start of this section to the start of the next section (that is, the offset from the first byte of this member to the first byte of the next section). Zero if the last section.	4 ASCII numeric characters
12	4	Base	Section Length -- The length of this section in bytes (including offset member above)	4 ASCII numeric characters “78 ”
16	8	Base	Section Name	8 ASCII characters “BASE ”
24	4	Base	system type	4 ASCII characters
28	10	Base	System serial number	10 ASCII characters: “pp-ssssss ”
38	4	Base	CoD capacity card CCIN	4 ASCII characters
42	10	Base	CoD capacity card serial number	10 ASCII characters “hh-hssssss”
52	16	Base	CoD capacity card unique ID	16 ASCII characters
68	4	Base	CoD resource identifier	4 ASCII characters
72	4	Base	Quantity of permanently activated resources	4 ASCII characters
76	4	Base	CoD sequence number	4 numeric ASCII characters

Table 98. Enhanced CoD Memory Capacity Info, Version 1 ***(Continued)***

Offset	Size in Bytes	Section	Description	Format
80	2	Base	CoD activation code entry check	1 byte hex check sum, 2 ASCII characters -- based on EBCDIC representation of previous 8 entries.
82	4	Base	Total CoD resources installed in system	4 numeric ASCII characters
On/Off Memory Resources				
0	4	OnOffMem	Decimal offset in bytes from the start of this section to the start of the next section (that is, the offset from the first byte of this member to the first byte of the next section). Zero if the last section.	4 ASCII numeric characters
4	4	OnOffMem	Section Length -- The length of this section in bytes (including offset member above)	4 ASCII numeric characters “67 “
8	8	OnOffMem	Section Name	8 ASCII characters “ONOFFMEM”
16	1	OnOffMem	On/Off CoD enabled	1 ASCII character '0' or '1'
17	1	OnOffMem	On/Off CoD active	1 ASCII character '0' or '1'
18	4	OnOffMem	On/Off CoD feature	4 ASCII characters
22	4	OnOffMem	On/Off CoD activated resources	4 ASCII numeric characters
26	4	OnOffMem	On/Off CoD sequence number	4 ASCII numeric characters
30	2	OnOffMem	On/Off CoD checksum	2 ASCII characters
32	4	OnOffMem	On/Off CoD resources requested	4 ASCII numeric characters
36	4	OnOffMem	On/Off CoD days requested	4 ASCII numeric characters
40	4	OnOffMem	On/Off CoD resource days expired	4 ASCII numeric characters
44	4	OnOffMem	On/Off CoD resource days remaining	4 ASCII numeric characters
48	4	OnOffMem	On/Off CoD counter	4 ASCII numeric characters
52	4	OnOffMem	On/Off standby resources available	4 ASCII numeric characters
56	1	OnOffMem	On/Off reserved byte	1 ASCII blank
57	4	OnOffMem	On/Off history of requested resource days	4 ASCII characters
61	1	OnOffMem	On/Off reserved byte	1 ASCII blank
62	4	OnOffMem	On/Off history of unreturned resource days	4 ASCII characters
66	1	OnOffMem	On/Off Memory Multiplier	1 ASCII numeric characters
Debit Memory Resources				
0	4	DebitMem	Decimal offset in bytes from the start of this section to the start of the next section (that is, the offset from the first byte of this member to the first byte of the next section). Zero if the last section.	4 ASCII numeric characters
4	4	DebitMem	Section Length -- The length of this section in bytes (including offset member above)	4 ASCII numeric characters “83 “

Table 98. Enhanced CoD Memory Capacity Info, Version 1 ***(Continued)***

Offset	Size in Bytes	Section	Description	Format
8	8	DebitMem	Section Name	8 ASCII characters “DEBITMEM”
16	1	DebitMem	Debit CoD enabled	1 ASCII character '0' or '1'
17	1	DebitMem	Debit CoD active	1 ASCII character '0' or '1'
18	4	DebitMem	Debit CoD feature	4 ASCII characters
22	4	DebitMem	Debit CoD activated resources	4 ASCII numeric characters
26	4	DebitMem	Debit CoD sequence number	4 ASCII numeric characters
30	2	DebitMem	Debit CoD checksum	2 ASCII characters
32	4	DebitMem	Debit CoD resources requested	4 ASCII numeric characters
36	12	DebitMem	Debit Reserved	12 ASCII blanks
48	4	DebitMem	Debit counter	4 ASCII characters
52	4	DebitMem	Debit standby resources available	4 ASCII numeric characters
56	1	DebitMem	Debit reserved byte	1 ASCII blank
57	4	DebitMem	Debit history of expired resource days	4 ASCII numeric characters
61	1	DebitMem	Debit reserved byte	1 ASCII blank
62	4	DebitMem	Debit history of unreturned resource days	4 ASCII characters
66	8	DebitMem	Extended total history of requested On/Off Memory GB days	8 ASCII characters
74	8	DebitMem	Extended total history of unreturned On/Off Memory GB days	8 ASCII characters
82	1	DebitMem	Debit reserved byte	1 ASCII blank
			Trial Memory Resources	
0	4	TrialMem	Decimal offset in bytes from the start of this section to the start of the next section (that is, the offset from the first byte of this member to the first byte of the next section). Zero if the last section.	4 ASCII numeric characters
4	4	TrialMem	Section Length -- The length of this section in bytes (including offset member above)	4 ASCII numeric characters “67 ”
8	8	TrialMem	Section Name	8 ASCII characters “TRIALMEM”
16	1	TrialMem	Trial CoD enabled	1 ASCII character '0' or '1'
17	1	TrialMem	Trial reserved	1 ASCII blank
18	4	TrialMem	Trial CoD feature	4 ASCII characters
22	4	TrialMem	Trial CoD activated resources	4 ASCII numeric characters
26	4	TrialMem	Trial CoD sequence number	4 ASCII numeric characters
30	2	TrialMem	Trial CoD checksum	2 ASCII characters

Table 98. Enhanced CoD Memory Capacity Info, Version 1 ***(Continued)***

Offset	Size in Bytes	Section	Description	Format
32	8	TrialMem	Trial reserved bytes	8 ASCII blanks
40	4	TrialMem	Trial days expired	4 ASCII numeric characters
44	4	TrialMem	Trial days remaining	4 ASCII numeric characters
48	14	TrialMem	Trial reserved bytes	14 ASCII blanks
62	4	TrialMem	Trial unreturned resources	4 ASCII numeric characters
66	1	TrialMem	Trial reserved byte	1 ASCII blank

R1–7.3.16.4.3–3. For the CoD option: The platform's *ibm,set-system-parameter* RTAS call specifying the Enhanced CoD Capacity Info, must not return a *Status* of 0 (Success); the expected return is a *Status* of -9002 (setting not allowed/authorized), however, under special cases a *Status* of -1 (Hardware Error) or one of the Busy or Extended Delay return *Status* values is allowed.

7.3.16.5 Restart Parameters

This section and its subsections describe parameters that govern the actions that the platform firmware takes upon a restart (that is, reboot) after an unintended termination.

7.3.16.5.1 partition_auto_restart Parameter

The partition_auto_restart parameter governs whether or not platform firmware attempts to restart a partition after an error which causes an abnormal partition termination. Neither a loss of external power without a UPS, nor a loss of external power and battery power with a UPS are examples of an error which causes abnormal partition termination. For terminations that involve only a specific partition (for example, a machine check), the partition_auto_restart parameter governs whether the partition restarts. For terminations that span the entire platform (for example, a checkstop), the platform may separately govern whether or not the entire platform restarts. If the platform does restart, however, partition_auto_restart determines whether or not an individual partition restarts.

R1–7.3.16.5.1–1. For the LPAR option with the System Parameters option: If the platform supports the partition_auto_restart system parameter, the platform must establish and maintain across boot (unless explicitly altered by a user) for each partition the one (1) byte parameter, the initial value (depending upon platform policy) is one of the following binary values:
0 - Do not automatically restart the partition
1 - Automatically restart the partition

R1–7.3.16.5.1–2. For the SMP option (non-LPAR) with the System Parameters option: If the platform supports the partition_auto_restart system parameter, the platform must establish and maintain across boot (unless explicitly altered by a user) one and only one (1) byte parameter, the initial value (depending upon platform policy) is one of the following binary values:
0 - Do not automatically restart the OS
1 - Automatically restart the OS

7.3.16.5.2 platform_auto_power_restart Parameter

The platform_auto_power_restart parameter governs whether or not platform firmware attempts to restart after power is restored following a power outage.

R1–7.3.16.5.2–1. For the System Parameters option: If the platform supports the platform_auto_power_restart system parameter, the platform must maintain across boot (unless explicitly altered by a user) one and only one platform wide value of the one (1) byte parameter having one of the following binary values:
0 - Do not automatically restart
1 - Automatically restart partitions that were active when external power was lost.

R1–7.3.16.5.2–2. For the LPAR option with the System Parameters option: If the platform supports the platform_auto_power_restart system parameter, the platform must provide the authority to set and read the platform_auto_power_restart system parameter to, at most, one partition at a time.

7.3.16.6 Remote Serial Port System Management Parameters

R1–7.3.16.6–1. For the LPAR option with the System Parameters option: If the platform supports any of the following system parameters: sp-remote-pon, sp-rb4-pon, sp-snoop-str, and sp-serial-snoop; the platform must grant authority to set and read the single platform wide values of the respective system parameters to only the partition owning the resource required to implement the function, such as a serial port, where the valid data for the parameters are specified in Table 93, “Defined Parameters,” on page 207.

R1–7.3.16.6–2. For the System Parameters option: The platform must support the sp-rb4-pon system parameter if and only if the sp-remote-pon system parameter is supported and implemented by using “Ring Indicate” of a serial port.

R1–7.3.16.6–3. For the LPAR option with the System Parameters option: Platforms that supports the sp-snoop-str system parameter must maintain one and only one platform wide NULL terminated ASCII string value of the parameter; granting authority to set and read the sp-snoop-str system parameter to, at most, one partition at a time.

R1–7.3.16.6–4. For the System Parameters option: To prevent return data truncation of the returned sp-snoop-str system parameter from the *ibm,get-system-parameter* RTAS call the caller must supply a buffer length sufficient to contain the two string length bytes plus the ASCII string and the terminating ASCII NULL.

R1–7.3.16.6–5. For the System Parameters option: The caller of the *ibm,get-system-parameter* RTAS call must supply a buffer length sufficient to contain the two string length bytes plus the ASCII string and the terminating ASCII NULL to prevent return data truncation of the returned sp-snoop-str system parameter.

R1–7.3.16.6–6. For the System Parameters option: The platform must supports both the sp-snoop-str and sp-serial-snoop system parameters if it supports either.

7.3.16.7 Surveillance Parameters

For the definition of the sp-sen, sp-sti, and sp-del parameters, see Section 7.3.5.2.1, “Surveillance,” on page 138.

R1–7.3.16.7–1. For the LPAR option with the System Parameters option: If the platform supports any of the following system parameters: sp-sen, sp-sti, or sp-del; the platform must grant authority to set and read the single platform wide one (1) byte values, where the decimal representation is defined in Table 93, “Defined Parameters,” on page 207, of the respective system parameters to only one partition at a time.

R1–7.3.16.7–2. For the System Parameters option: If the platform supports any of the sp-sen, sp-sti or sp-del system parameters, it must support then all.

7.3.16.8 Call Home Parameter

This parameter is used to provide input concerning certain call home values used when a call home function is provided. The data for the parameter is an ASCII string which provides additional information

R1–7.3.16.8–1. For the LPAR option with the System Parameters option: If the platform supports the sp-call-home parameter, platform must grant authority to set and read the single platform wide value of the system parameter at any time to only one partition; where the data for the parameter is an ASCII string in the form <String_name1>=<string><ASCII NULL><String_name2>=<string><ASCII NULL>....<String_nameN>=<string><ASCII NULL><ASCII NULL> with string names defined as per Table 99, “sp-call-home Strings,” on page 223.

R1–7.3.16.8–2. For the System Parameters option: The caller of the *ibm,get-system-parameter* RTAS call must supply a buffer length sufficient to contain the maximum possible ASCII string returned, including the two ASCII NULLs where Table 99, “sp-call-home Strings,” on page 223 indicates the maximum length of the data for each substring that comprises the sp-call-home data, to prevent return data truncation of the returned sp-call-home system parameter.

R1–7.3.16.8–3. For the System Parameters option: If the platform supports the sp-call-home parameter, the platform must provide the sp-call-home parameter value defaults listed in Table 99, “sp-call-home Strings,” on page 223 prior to any *ibm,set-system-parameter* RTAS call.

Table 99. sp-call-home Strings

String_Name	Default	Range	Maximum Characters in String Data	Description
sp-rt-s<N>	NULL		20	Retry string for serial port <N>
sp-ic-s<N>	NULL		12	Protocol interdata block delay (*IC) for serial port <N>
sp-to-s<N>	NULL		12	Protocol time out (*DT) for serial port <N>
sp-cd-s<N>	NULL		12	Call Delay (*CD) for serial port <N>
sp-connect-s<N>	NULL		12	Connect (*CX) for serial port <N>
sp-disconnect-s<N>	NULL		12	Disconnect (*DX) for serial port <N>
sp-condout-s<N>	NULL		12	Call-out condition (*C0) for serial port <N>
sp-condwait-s<N>	NULL		12	Call-wait (*C0) for serial port <N>
sp-condin-s<N>	NULL		12	Call-in condition (*C1) for serial port <N>
sp-waitcall-s<N>	NULL		12	Wait call (*WC) for serial port <N>
sp-page-s<N>	NULL		20	Describes how to Page a beeper for serial port <N>
sp-diok-s<N>	off	on,off	4	Serial Port <N> Call-in (Dial in authorized on the port)
sp-dook-s<N>	off	on,off	4	Serial Port <N> Call-out (Dial out authorized on the port)
sp-dookc	off	on,off	4	Call-out before restart (Dial out for system crash using authorized serial port)
sp-ls-s<N>	9600	300, 600, 1200, 2000, 2400, 3600, 4800, 7200, 9600, 19200, 38400	6	S<N> line speed
sp-modemf-s<N>	NULL		120	Filename of the last modem file used to configure modem parameters
sp-phsvc	20 blank characters	20 characters max	20	Service Center Telephone Number (*PS)
sp-phadm	20 blank characters	20 characters max	20	Customer Administration Center Telephone Number (*PH)

Table 99. sp-call-home Strings

String_Name	Default	Range	Maximum Characters in String Data	Description
sp-pager	20 blank characters	20 characters max	20	Digital Pager Telephone Number
sp-phsys	20 blank characters	20 characters max	20	Customer System Telephone number (*PY)
sp-vox	20 blank characters	20 characters max	20	Customer Voice telephone number (*PO)
sp-acct	12 blank characters	12 characters max	12	Customer Account Number (*CA)
sp-cop	first	first, all	6	Call-out policy (first/all) - numbers to call in case of failure
sp-retlogid	12 blank characters	12 characters max	12	Customer RETAIN Login Userid (*LI)
sp-retpw	16 blank characters	16 characters max	12	Customer RETAIN Login password (*PW)
sp-rto	120	>1	12	Remote Timeout (in seconds) (*RT)
sp-rlat	2	> 1	12	Remote Latency (in seconds) (*RL)
sp-rn	2	0 or any positive number	12	Number of retries (while busy (*RN))
sp-sysname	15 blank characters	15 characters max	15	System Name (system administrator aid)

Notes:

1. <N> is substituted with a modem number, i.e. 1 or 2.
2. NULL as a default indicates that the string is given a name, but no value, e.g. sp-modemf-s=

7.3.16.9 Current Flash Image Parameter

In systems with storage for more than one Flash image, the sp-current-flash-image parameter indicates which Flash image is currently being used by the service processor. This is typically the Flash image used at the last boot.

R1–7.3.16.9–1. For the LPAR option with the System Parameters option: Platforms that supports the sp-current-flash-image system parameter, must authorize all partitions to get the single platform wide one (1) byte value of the system parameter, whose decimal representation is defined in Table 93, “Defined Parameters,” on page 207.

R1–7.3.16.9–2. For the System Parameters option: Platforms that supports the sp-current-flash-image system parameter must support the *ibm,manage-flash-image* RTAS call.

7.3.16.10 Platform Dump Max Size Parameter

This parameter indicates the size (in bytes) needed for dumps returned from the *ibm,platform-dump* RTAS function.

R1–7.3.16.10–1. For the Platform Dump option: If the *ibm,platform-dump* RTAS call is authorized for the partition, the platform must authorize the partition to get the platform-dump-max-size system parameter; where the value returned must indicate the sum (in bytes) of the maximum size of each unique platform dump type that the *ibm,platform-dump* RTAS call could return.

Programming Note: The intent of platform-dump-max-size is for the platform to specify, in advance, the sum of the maximum sizes of all the unique dump types that it can generate. This is to allow the OS to reserve space for one log of each type. In the case of any change in the value of this parameter, the platform may generate a Platform

Event Log entry announcing the change in the maximum size, and specifying the new size in the IO Events Section. This entry, when generated, is then returned by the *event-scan* RTAS call.

7.3.16.11 Storage Preservation Option System Parameters

The epow3-quiesce-time system parameter contains the time granted to the current instance of a client program to perform quiesce activities in preparation for a memory preservation boot. This quiesce time is the time used by the client program to do such things as quiesce and power off I/O not needed for memory preservation boot processing, in order to conserve batteries. A client program utilizing the Storage Preservation option, upon completion of quiesce activities, requests a reboot. The platform, upon seeing an EPOW class 3 condition, and if both the memory-preservation-boot-time and epow3-quiesce-time system parameters are non-zero, starts a timer with an initial value equal to the epow3-quiesce-time. If the timer expires before the client program performs a reboot, the platform forces a reboot of the client program.

The memory-preservation-boot-time system parameter contains the time granted to the rebooted instance of the partition to perform the saving of preserved memory. The client program, upon completion of the saving of preserved memory, requests a shutdown. The platform, upon initiation of a memory preservation boot starts a timer with an initial value equal to the memory-preservation-boot-time, providing the value of the memory-preservation-boot-time parameter is non-zero. If the timer expires before the client program performs a shutdown, the platform forces a shutdown of the client program.

Thus, the platform uses the memory-preservation-boot-time system parameter as a policy attribute. If the client program has set the value of this parameter to a non-zero value, then the memory preservation boot timers are enabled. If the memory-preservation-boot-time parameter is zero (independent of the epow3-quiesce-time setting), the platform does not initiate the memory preservation boot timers.

To use the memory preservation boot timers, the client program registers its LMBs for preservation and sets the memory-preservation-boot-time via the *ibm,set-system-parameter* RTAS call. If an EPOW class 3 is sent to a client program and the client program has set its memory-preservation-boot-time parameter, then the platform starts the timer for epow3-quiesce-time. The client program on reboot uses the *get-sensor* RTAS call (to detect EPOW condition) and the **`"ibm,preserved-storage"`** property in the device tree to drive memory preservation processing as necessary. The values of memory-preservation-boot-time and epow3-quiesce-time prior to being set for a client program are 0. These system parameters are persisted, as are all system parameters.

R1–7.3.16.11–1. For the Storage Preservation option: The platform must implement the memory-preservation-boot-time and epow3-quiesce-time system parameters and must set their initial values to 0.

R1–7.3.16.11–2. For the Storage Preservation option: If the memory-preservation-boot-time system parameter is non-zero for a client program and if the platform delivers an EPOW class 3 indication to the client program, the platform must do all of the following:

a. Upon delivering the EPOW class 3 to the client program, if the epow3-quiesce-time system parameter is non-zero, then set a timer based on the client program's epow3-quiesce-time system parameter and force a reboot of the client program on timer expiration, if the client program does not request a reboot itself before the timer expires.

b. Upon initiation of the memory preservation boot, set a timer based on the client program's memory-preservation-boot-time system parameter and on timer expiration, force a shutdown of the client program if the client program does not request a shutdown itself before the timer expires.

7.3.16.12 SCSI Initiator Identifier System Parameters

Certain physical SCSI IOAs maintain their previous settings for SCSI initiator identifier, while others require that the platform set this value during I/O adapter initialization. Since the initialization of I/O adapters in the boot path is done by firmware, a method is required for the OS to inform the platform firmware of such settings. Given that an OS owns

a slot, and that slot contains a supported SCSI I/O adapter, the OS may use the *ibm,set-system-parameter* RTAS call specifying SCSI Initiator Identifier system parameters to instruct the firmware how to initialize the I/O adapter to ensure that it does not conflict with other SCSI initiators on the same bus. The *ibm,get-system-parameter* RTAS call is used to verify the SCSI Initiator Identifier system parameters value for any OS owned slot.

When *ibm,set-system-parameter* is called specifying SCSI Initiator Identifier system parameters, the buffer contains the standard two byte length field plus two NULL terminated strings. The first string contains the location code of an I/O Adapter's SCSI bus connector, and the second string contains one of the decimal values 0-15 representing the value of the SCSI Initiator Identifier that the platform's firmware is to use to initialize the SCSI controller for that bus.

When *ibm,get-system-parameter* is called specifying SCSI Initiator Identifier system parameters, the buffer contains the standard two byte length field plus a NULL terminated string that contains the location code of an I/O Adapter's SCSI bus connector. Upon successful return, the buffer contains the standard two byte length field plus two NULL terminated strings. The first string contains the location code of the I/O Adapter's SCSI bus connector, and the second string contains one of the decimal values 0-15 representing the value of the SCSI Initiator Identifier that the platform's firmware is to use to initialize the SCSI controller for that bus.

Implementation Note: For IOAs that have multiple connectors per bus, the location code specifies the connector for the external bus.

Interaction between SCSI Initiator Identifier system parameters and DR operations produce unique situations. The platform maintains only the latest SCSI Initiator Identifier set for any given location code. On DR operations, the value is normally retained until the IOA owner explicitly changes it. If a DR operation replaces the original IOA with a different type of IOA, such that the previously set SCSI Initiator Identifier system parameters no longer make sense (IOA is not a supported SCSI adapter or the connector location codes do not match), the platform firmware clears the SCSI Initiator Identifier system parameters for the location code and performs the platform default IOA initialization.

R1–7.3.16.12–1. For the SCSI Initiator Identifier System Parameters option: When *ibm,set-system-parameter* is called specifying SCSI Initiator Identifier system parameters, RTAS must return *Status* of -3 (Parameter error) on any of the following conditions:

a. The binary value of the first two bytes in the buffer, plus 2, is greater than the buffer length parameter.

b. The buffer length parameter is greater than 1026.

c. The N bytes of buffer contents (N being the binary value of the first two buffer bytes) does not contain two NULL terminated strings.

d. The contents of the first NULL terminated buffer string does not match the format of a valid platform location code.

e. The contents of the second NULL terminated buffer string does not contain a decimal value in the range of 0 to 15.

R1–7.3.16.12–2. For the SCSI Initiator Identifier System Parameters option: When *ibm,set-system-parameter* is called specifying SCSI Initiator Identifier system parameters, and the request successfully passes the Requirements of R1–7.3.16.12–1, the first NULL terminated buffer string must contain a valid formatted platform location code for a currently installed slot owned by the calling OS, or the platform must return “Not authorized” *Status*.

R1–7.3.16.12–3. For the SCSI Initiator Identifier System Parameters option: When *ibm,set-system-parameter* is called specifying SCSI Initiator Identifier system parameters, and the request successfully passes the Requirements of R1–7.3.16.12–2, the first NULL terminated buffer string must contain a valid formatted platform location code for a SCSI bus connector of a supported SCSI I/O adapter currently installed in a slot owned by the calling OS, or the platform must return “Parameter Error” *Status*.

R1–7.3.16.12–4. For the SCSI Initiator Identifier System Parameters option: When *ibm,set-system-parameter* is called specifying SCSI Initiator Identifier system parameters, and the request successfully passes the Requirements of R1–7.3.16.12–3, the firmware must record the value supplied in the second NULL terminated buffer string for use in initializing the SCSI initiator identifier of the SCSI I/O adapter contained in the slot specified by the first NULL terminated buffer string and return a *Status* of 0 (success) (except in the case of hardware errors or busy conditions).

R1–7.3.16.12–5. For the SCSI Initiator Identifier System Parameters option: When *ibm,get-system-parameter* is called specifying SCSI Initiator Identifier system parameters, RTAS must return a *Status* of -3 (Parameter error) on any of the following conditions:

a. The binary value of the first two bytes in the buffer, plus 2, is greater than the buffer length parameter.

b. The buffer length parameter is greater than 1026.

c. The N bytes of buffer contents (N being the binary value of the first two buffer bytes) does not contain one NULL terminated string.

d. The contents of the NULL terminated buffer string does not match the format of a valid platform location code.

R1–7.3.16.12–6. For the SCSI Initiator Identifier System Parameters option: When *ibm,get-system-parameter* is called specifying SCSI Initiator Identifier system parameters, and the request successfully passes the Requirements of R1–7.3.16.12–5, the NULL terminated buffer string must contain a valid formatted platform location code for a currently installed slot owned by the calling OS, or the platform must return “Not authorized” *Status*.

R1–7.3.16.12–7. For the SCSI Initiator Identifier System Parameters option: When *ibm,get-system-parameter* is called specifying SCSI Initiator Identifier system parameters, and the request successfully passes the Requirements of R1–7.3.16.12–6, the NULL terminated buffer string must contain a valid formatted platform location code for a SCSI bus connector of a supported SCSI I/O adapter currently installed in a slot owned by the calling OS, or the platform must return a *Status* of -3 (parameter error).

R1–7.3.16.12–8. For the SCSI Initiator Identifier System Parameters option: When *ibm,get-system-parameter* is called specifying SCSI Initiator Identifier system parameters, and the request successfully passes the Requirements of R1–7.3.16.12–7, the firmware must:

a. Increase the value contained in the first two bytes of the buffer to cover both the length of the location code NULL terminated string and a NULL terminated string representing the decimal value that the platform uses to initialize the SCSI initiator identifier of the SCSI I/O adapter contained in the slot specified by the first NULL terminated buffer string.

b. If there is room in the buffer, append the NULL terminated string representing the decimal value that the platform uses to initialize the SCSI initiator identifier of the SCSI I/O adapter contained in the slot specified by the first NULL terminated buffer string.

c. Return a *Status* of 0 (success) (except in the case of hardware errors or busy conditions).

R1–7.3.16.12–9. For the SCSI Initiator Identifier System Parameters option: When the platform firmware initializes an IOA and a SCSI Initiator Identifier system parameter is set for that IOA's slot location code, and the SCSI Initiator Identifier system parameter is incompatible with the currently installed IOA (IOA is not a supported SCSI adapter or the connector location codes do not match a SCSI bus connector for that IOA), the platform must clear the incompatible SCSI Initiator Identifier system parameter and proceed to initialize the IOA using platform default behaviors.

7.3.16.13 CoD Options

The CoD Options system parameter allows specification of various CoD options.

R1–7.3.16.13–1. When *ibm,get-system-parameter* is called specifying the CoD Options system parameter, the first two bytes of the value returned must contain the full length of the parameter data, including the length of the NULL. The two byte binary length field is followed by a variable of the form "keyword" followed by "=" followed by "value" and terminated by a semicolon (";"), where the contents of "value" must be an ASCII printable character string.

R1–7.3.16.13–2. The corresponding keyword and values for the CoD Options parameter are defined in Table 100, "CoD Options System Parameter Keyword and Values," on page 228.

Table 100. CoD Options System Parameter Keyword and Values

Keyword	Permitted Values	Definition
LPoptions	yes, no	no: The platform does not support the Low Priced adapters and devices. yes: The platform supports the Low Priced adapters and devices. Absence of the keyword is the same as the keyword with the value of "yes".

7.3.16.14 Platform Error Classification

The Platform Error Classification system parameter specifies whether the OS should process platform reported errors as informational errors as opposed to service actionable events.

R1–7.3.16.14–1. When *ibm,get-system-parameter* is called specifying the Platform Error Classification system parameter, the platform must return a value of `"1"` if all errors returned in *event-scan*, *check-exception*, *rtas-last-error* and *ibm,slot-error-detail* calls should be treated as informational errors in the sense that they not be reported by service applications as service actionable events and otherwise must return a value of `"0"`.

Programming Note: Service applications within an operating system may obtain information about platform errors and take service actions (such as reporting the errors to a call center or other error aggregation point) based on errors logged. Service applications running in multiple partitions, each receiving platform error events, may all report the same error to an aggregation point causing duplicated error reports. To eliminate this duplication, a platform might choose to log errors to only one partition in a system. That, however, would leave an incomplete error record in individual partition and eliminate notifications that each partition OS should be aware of (such as EPOW events).

To allow platform errors to be reported to an OS, but prevent the forwarding of the errors as service actionable events to an error aggregation point, the Platform Error Classification system parameter may be set to a value of 1.

The OS should not change how it logs an error based on this parameter, nor should the OS change any error severity associated with the log based on the parameter. Rather it is left to service applications to query the system parameter and take actions based on it.

7.3.16.15 Firmware Boot Options

The Firmware Boot Options system parameter allows specification of various firmware boot settings.

R1–7.3.16.15–1. When *ibm,get-system-parameter* is called specifying the Firmware Boot Options system parameter, the first two bytes of the value returned must be binary and must contain the full length of the parameter data, including the length of the NULL, and the field following length field must be a variable of the form:

"keyword" followed by "=" followed by "value" and terminated by a semicolon (";"), where the contents of "value" must be an ASCII printable character string.

R1–7.3.16.15–2. When *ibm,set-system-parameter* is called specifying the Firmware Boot Options system parameter, the first two bytes of the buffer must be binary and must contain the full length of the parameter data, including the length of the NULL, and the field following length field must be a variable of the form: "keyword" followed by "=" followed by "value" and terminated by a semicolon (";"), where the contents of "value" must be an ASCII printable character string, and if the caller is not authorized to adjust at least one of the specified keywords, the call must return with a status of -9002.

R1–7.3.16.15–3. Keyword and values for the Firmware Boot Options parameter must be as defined in Table 101, "Firmware Boot Options System Parameter Keywords and Values," on page 229.

Table 101. Firmware Boot Options System Parameter Keywords and Values

Keyword	Permitted Values	Definition
PlatformBootSpeed	fast, slow	fast: The platform will perform a minimal set of hardware tests before loading the OS. slow: The platform will perform a comprehensive set of hardware tests before loading the OS. Absence of the keyword implies the platform does not support an alterable boot speed.

7.3.16.16 Platform Processor Diagnostics Options

The platform-processor-diagnostics-run-mode system parameter allows the operating system to query or control how platform run-time processor diagnostics are executed by the platform. Provision is made by this parameter for the platform to execute run-time diagnostic tests to verify various processor functions. These diagnostics tests typically would be performed by the hypervisor against each processor in the system.

R1–7.3.16.16–1. When *ibm,get-system-parameter* is called with the platform-processor-diagnostics-run-mode token, the platform must return a one-byte parameter indicating the current run-mode of platform processor diagnostics as one of the following:

a. 0 = disabled: indicates that the platform will not run processor run-time diagnostics.

b. 1 = staggered: indicates that the platform is set to run processor diagnostics on each processor on a periodic basis, but not attempt to schedule the tests for all processors at the same time. The frequency at which the tests will run are defined by the platform.

c. 2 = immediate: indicates that the platform is currently in the processor of running diagnostics against the processors in a system on a non-staggered basis, either as a result of an "immediate" or "periodic" setting.

d. 3 = periodic: indicates that the platform is scheduled to run diagnostics against all the processors in the system at a specific time scheduled by the platform.

R1–7.3.16.16–2. When *ibm,set-system-parameter* is called specifying the platform-processor-diagnostics-run-mode token, the one-byte parameter passed must indicate the run-mode of platform periodic diagnostics desired as one of the following:

a. 0 = disabled: indicates that the platform should not run any processor run-time diagnostics. Any currently running diagnostics will be terminated.

b. 1 = staggered: indicates that the platform should run diagnostics periodically against each processor in the system, but not attempt to schedule the tests for all processors at the same time. The frequency at which the tests will run are defined by the platform.

c. 2 = immediate: indicates that the platform should immediately begin the process of running processor diagnostics on all of the processors in the system, This setting only temporarily overrides the setting of "disabled", "staggered" or "periodic" and the platform will revert to the last setting of "disabled", "staggered" or "periodic" once the immediately run diagnostics are complete.

Implementation Notes: To prevent conflicts in the setting of the run-mode, the platform should only support this parameter for one partition in a running system. The options may also be set by the platform. The last value set will take precedent over any previous settings.

7.3.16.17 Processor Module Information

The Processor Module Information system parameter allows transferring of certain processor module information from the platform to the OS. The information in the parameter is global for the platform and encompasses all resources on the platform, not just those available to the partition, and the *ibm,get-system-parameter* will never return a *Status* of -9002 (Not Authorized). This parameter is read-only.

R1–7.3.16.17–1. For the LPAR option with the System Parameters option: If the platform supports the Processor Module Information system parameter, then it must provide the following information in the parameter, and the information returned for every partition must be the same, with all the resources of the platform encompassed:

2 byte binary number (N) of module types followed by N module specifiers of the form:
2 byte binary number (M) of sockets of this module type
2 byte binary number (L) of chips per this module type
2 byte binary number (K) of cores per chip in this module type.

R1–7.3.16.17–2. For the LPAR option with the System Parameters option: For the Processor Module Information system parameter, the *ibm,get-system-parameter* RTAS call must never return a *Status* of -9002 (Not Authorized), and the *ibm,set-system-parameter* RTAS call must always return a Status of -9002 (Setting not allowed/authorized).

7.3.16.18 Cede Latency Settings Information

The Cede Latency Settings Information system parameter informs the OS of the maximum latency to wake up from the various platform supported processor sleep states that it might employ for idle processors. The information in the parameter is global for the platform and encompasses all processors on the platform, and the *ibm,get-system-parameter* will never return a *Status* of -9002 (Not Authorized). This parameter is read-only. As the architecture evolves, the number of fields per record are likely to increase, calling software should be written to handle fewer fields (should it find itself running on a platform supporting an older version of the architecture) and ignore additional fields (should it find itself running on a platform supporting a newer version of the architecture). Due to partition migration, the support for the cede latency setting system parameter, the number of supported cede latency settings (and thus the number of reported records) and the number of fields reported per record might change from call to call; calling software should be written to handle this variability

R1–7.3.16.18–1. For the PEM option with the System Parameters option: If the platform supports the cede latency settings information system parameter it must provide the following information in the NULL terminated parameter string:

a. The first byte is the binary length "N" of each cede latency setting record minus one (zero indicates a length of 1 byte)

b. For each supported cede latency setting a cede latency setting record consisting of: The first "N" bytes of Table 102, "Byte definitions within a cede latency setting record," on page 231.

Table 102. Byte definitions within a cede latency setting record

Order of fields within a record	Field Length	Values	Comments
Cede Latency Specifier Value	1	Binary Values 0-255	Records in ascending cede latency specifier value order with no holes.
Maximum wakeup latency in time base ticks	8	0x0000000000000000 – 0xFFFFFFFFFFFFFFFF	
Responsive to external interrupts	1	Binary True/False	

R1–7.3.16.18–2. For the PEM option with the System Parameters option: For the cede latency specifier system parameter, the *ibm,get-system-parameter* RTAS call must never return a *Status* of -9002 (Not Authorized), and the *ibm,set-system-parameter* RTAS call must always return a *Status* of -9002 (Setting not allowed/authorized).

7.3.16.19 Target Active Memory Compression Factor

The target active memory compression factor system parameter informs the OS of the target memory capacity increase the customer expects to achieve due to active memory compression. The factor is expressed in whole percentage with the minimum value of 100 and the maximum value of 1000.

The *ibm,get-system-parameter* for parameter token 46 will never return a *Status* of -9002 (Not Authorized). This parameter is read-only.

R1–7.3.16.19–1. For the Active Memory Compression option with the System Parameters option: For the Target Active Memory Compression Factor system parameter, the *ibm,get-system-parameter* RTAS call must never return a Status of -9002 (Not Authorized).

R1–7.3.16.19–2. For the Active Memory Compression option with the System Parameters option: If the Active Memory Compression option is enabled for the partition, the platform must provide in response to the *ibm,get-system-parameter* for parameter token 46 the two byte target active memory compression factor in binary format in the range (0x0064 -- 0x03E8) (equivalent to 100 -- 1000 decimal).

R1–7.3.16.19–3. For the Active Memory Compression option with the System Parameters option: If the Active Memory Compression option is disabled for the system/partition, the platform must provide in response to the *ibm,get-system-parameter* for parameter token 46 the two byte value 0x0000.

R1–7.3.16.19–4. For the Active Memory Compression option with the System Parameters option: For the target active memory compression factor system parameter, the *ibm,set-system-parameter* RTAS call must always return a Status of -9002 (Setting not allowed/authorized).

7.3.16.20 Performance Boost Modes Vector

A variety of platform dependent configuration modes might result in a boost in platform computational capacity. The *ibm,get-system-parameter* through the performance boost modes vector system parameter communicates to the client program which of these modes are available on the specific platform, which of these modes the client program may enable/disable, and which ones are active.

The performance boost mode vectors are 32 bytes (256 bits) long. Each bit position within the performance boost mode vector corresponds to a specific function as specified in Table 103, “Performance Boost Modes Vector Bits Defini-

tions," on page 232. The first defined boost mode is assigned to the highest order bit position. As new boost modes are defined, they are assigned to sequential lower order vector bit positions.

Given that the second version of the vector from the *ibm,get-system-parameter* RTAS call (specifying which modes may be enabled/disabled by the client program) is non-zero, the platform supports calling the *ibm,set-system-parameter* RTAS call specifying the performance boost modes vector token. The *ibm,set-system-parameter* RTAS call specifying the performance boost modes vector token takes a single vector as input.

Table 103. Performance Boost Modes Vector Bits Definitions

Bit Position (1 based ordinal)	Definition
1	Extended Cache Option
2-- 256	Reserved

R1–7.3.16.20–1. For the Performance Boost Modes option: The platform must implement the System Parameters option.

R1–7.3.16.20–2. For the Performance Boost Modes option: The 96 byte report returned by *ibm,get-system-parameter* for parameter token 47 must consist of three 32 byte bit vectors as defined by Table 103, "Performance Boost Modes Vector Bits Definitions," on page 232.

R1–7.3.16.20–3. For the Performance Boost Modes option: The first 32 byte bit vector returned by *ibm,get-system-parameter* for parameter token 47 must contain 1s in the bit positions define by Table 103, "Performance Boost Modes Vector Bits Definitions," on page 232 for the performance boost modes that are both supported by the platform and authorized for the caller (by means outside of the scope of LoPAPR).

R1–7.3.16.20–4. For the Performance Boost Modes option: The second 32 byte bit vector returned by *ibm,get-system-parameter* for parameter token 47 must contain 1s in the bit positions define by Table 103, "Performance Boost Modes Vector Bits Definitions," on page 232 for the performance boost modes that are both represented in the first vector and may be enabled/disabled by the caller through the *ibm,set-system-parameter* using parameter token 47.

R1–7.3.16.20–5. For the Performance Boost Modes option: The third 32 byte bit vector returned by *ibm,get-system-parameter* for parameter token 47 must contain 1s in the bit positions define by Table 103, "Performance Boost Modes Vector Bits Definitions," on page 232 for the performance boost modes that are both represented in the first vector and are enabled either by default or by the caller through the *ibm,set-system-parameter* using parameter token 47.

R1–7.3.16.20–6. For the Performance Boost Modes option: If the *ibm,get-system-parameter* for parameter token 47 communicated that the client program has the ability to enable/disable one or more of the boost modes, then the platform must support the performance boost modes vector token for ibm,set-system-parameter.

R1–7.3.16.20–7. For the Performance Boost Modes option: If no boost modes can be enabled/disabled then a call to ibm,set-system-parameter specifying the boost modes vector token must return either:

- "System parameter not supported" as indeed the implementation need not code support for the token if no mode setting is supported.
- "Setting not allowed/authorized" if the implementation supports setting boost modes but the caller is not authorized to do so.

R1–7.3.16.20–8. For the Performance Boost Modes option: If any input vector to the *ibm,set-system-parameter* RTAS for parameter token 47 is a one and does not correspond to a bit that is a one in the second version of

the vector returned by the *ibm,get-system-parameter* RTAS for parameter token 47 the ibm,set-system-parameter RTAS must return parameter error.

R1–7.3.16.20–9. For the Performance Boost Modes option: If the corresponding bit that was a one in the second version of the vector returned by the *ibm,get-system-parameter* RTAS for parameter token 47 is a one in the input vector to the *ibm,set-system-parameter* RTAS for parameter token 47 then upon successful return that corresponding boost mode must be enabled.

R1–7.3.16.20–10. For the Performance Boost Modes option: If the corresponding bit that was a one in the second version of the vector returned by the *ibm,get-system-parameter* RTAS for parameter token 47 is a zero in the input vector to the *ibm,set-system-parameter* RTAS for parameter token 47 then upon successful return that corresponding boost mode must be disabled.

R1–7.3.16.20–11. For the Performance Boost Modes option: To properly awake from partition suspension and handle dynamic reconfiguration, the client program must be prepared to handle changes in the bit settings within the bit vectors reported by the *ibm,get-system-parameter* RTAS for parameter token 47.

R1–7.3.16.20–12. For the Performance Boost Modes option: Since it is expected that bit positions define by Table 103, “Performance Boost Modes Vector Bits Definitions,” on page 232 will expand over time, to avoid firmware level compatibility issues, the client program must ignore bit settings within the bit vectors reported by the *ibm,get-system-parameter* RTAS for parameter token 47 beyond those defined when the client pr gram was designed.

7.3.16.21 Universally Unique IDentifier

The Universally Unique IDentifier (UUID) option provides each partition with a Universally Unique Identifier that is persisted by the platform across partition reboots, reconfigurations, OS reinstalls, partition migration, hibernation etc. The UUID is a 16 byte string of format fields and random bits as defined in Table 104, “UUID Format,” on page 233. The random bits are generated in an implementation dependent manner to achieve a projected probability of collision of not greater than one in 2**60.

Table 104. UUID Format

Field	Byte:Bit	Size (Bits)	Values
Version	0:0	1	0: Initial Version 1: Reserved
Random Bits	0:1 thru 5:7	47	Random Bits
Generation Method	6:0-3	4	0b0000 Never Used 0b0100 Random Generated All other values are reserved
Random Bits	6:4 - 7:7	12	Random Bits
Variant	8:0-1	2	0b10 DCE Variant UUID All other values are reserved
Random Bits	8:2 - 15:7	62	Random Bits

R1–7.3.16.21–1. For the UUID option with the System Parameters option: For the UUID system parameter, the *ibm,get-system-parameter* RTAS call must never return a Status of -9002 (Not Authorized).

R1–7.3.16.21–2. For the UUID option with the System Parameters option: If the UUID option is enabled for the partition, the platform must provide in response to the *ibm,getsystem-parameter* for parameter token 48 the calling partition unique 16 byte sting as described in Table 104, “UUID Format,” on page 233.

R1–7.3.16.21–3. For the UUID option with the System Parameters option: If the UUID option is disabled for the system/partition, the platform must provide in response to the *ibm,get-system-parameter* for parameter token 48 the two byte value 0x0000.

R1–7.3.16.21–4. For the UUID option with the System Parameters option: For the UUID system parameter, the *ibm,set-system-parameter* RTAS call must always return a Status of -9002 (Setting not allowed/authorized).

7.4 *ibm,get-indices* RTAS Call

The RTAS function *ibm,get-indices* is used to obtain the indices and location codes for a specified indicator or sensor token. It allows for obtaining the list of indicators and sensors dynamically and therefore assists in any Dynamic Reconfiguration operation that involves indicators and sensors being added or deleted from the platform (unlike the **/rtas** node **"**<*vendor*>**,indicator-**<*token*>**"**, **"**<*vendor*>**,sensor-**<*token*>**"**, and **"ibm,environmental-sensor"** properties). This call also allows discontiguous indices for a particular indicator or sensor type (unlike the **"rtas-indicators"**, **"rtas-sensors"**, and **"ibm,environmental-sensor"** properties).

This RTAS call is not used for DR indicators (9001, 9002, and 9003) or DR sensors (9003). See the following sections in the DR chapter for more information: Section 13.5.3.3, “get-sensor-state,” on page 366 and Section 13.5.3.4, “set-indicator,” on page 367.

It may require several calls to the *ibm,get-indices* RTAS routine to get the entire list of indicators or sensors of a particular type. Each call may specify a different work area.

The OS may not interleave calls to *ibm,get-indices* for different indicator or sensor types. Other standard RTAS locking rules apply.

R1–7.4.0–1. For all DR options: The RTAS function *ibm,get-indices* must implement the argument call buffer defined by Table 105, “ibm,get-indices Argument Call Buffer,” on page 234.

Table 105. *ibm,get-indices* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,get-indices*
	Number Inputs	5
	Number Outputs	2
	Indicator or Sensor	0: indicator of given type 1: sensor of given type
	Indicator Type	Indicator or sensor type (for example, 9006, 9007)
	Work Area Address	Address of work area
	Work Area Size	Size of work area
	Starting Number	Integer representing first indicator number to return
Out	*Status*	-1: Hardware error -3: Indicator type not supported -4: Optional: Indicator list changed, start again 0: Success 1: More data available; call again 990x:Extended Delay where x is a number 0-5 (see text below)
	Next Starting Number	Integer to use as the Starting Number parameter on the next call, or 1 if no more calls are required

When the 990x *Status* is returned, it is suggested that software delay for 10 raised to the x milliseconds (where x is the last digit of the 990x return code), before calling *ibm,get-indices* with the same Starting Number and Indicator Type. However, software may issue the *ibm,get-indices* call again either earlier or later than this.

R1–7.4.0–2. The OS must not interleave calls to *ibm,get-indices* for different indicator or sensor types.

R1–7.4.0–3. On the first call to get a particular *Indicator Type*, the caller must provide a Starting Number of 1 (32-bit integer)

R1–7.4.0–4. When *ibm,get-indices* is called with a Starting Number of 1, firmware must refresh any stale data in previously cached firmware buffers for that indicator (for example, data made stale by a Dynamic Reconfiguration operation).

R1–7.4.0–5. When calling *ibm,get-indices* with a Starting Number of 1, a previously returned *Next Starting Number* value must be discarded.

R1–7.4.0–6. Optionally, if firmware detects a change in the indicator list before the entire list is returned, the *ibm,get-indices* call must return a -4 and the caller must start again with a Starting Number of 1.

R1–7.4.0–7. The return data format in the work area for all sensors and indicators must be as follows:

- Number Returned: 32-bit integer representing the number of indicator indices returned on this call
- Sets of (32-bit integer index, 32-bit integer length of location code including NULLs, location code string (NULL terminated and padded to nearest 4 byte boundary)), one set per indicator or sensor, with the number of sets indicated by the first integer in this work buffer

R1–7.4.0–8. If the *Status* returned is 1 (more data available, call again), then the caller must call *ibm,get-indices* again with the *Starting Number* parameter set to the Next Starting Number integer from the previously returned buffer.

R1–7.4.0–9. The *ibm,get-indices* RTAS call must return the *Status* value of -3 for the following conditions:

a. Indicator type not supported

b. No indicators of specified Indicator Type available to caller

R1–7.4.0–10. If the *ibm,get-indices* RTAS call returns a *Status* of anything other than 0 or 1 is returned, the caller must consider that the contents of the work area is not defined.

R1–7.4.0–11. The work area specified in the *ibm,get-indices* RTAS call argument buffer must be contiguous in logical real memory and must reside below 4GB.

R1–7.4.0–12. The *ibm,get-indices* RTAS call must only return the indicator or sensor indices to which the caller has authorized access at the time of the call.

R1–7.4.0–13. The *ibm,get-indices* RTAS call must make no assumptions about the contents of the work area on the beginning of the call.

R1–7.4.0–14. When the platform supports the *ibm,get-indices* RTAS call, the device tree must include the **"ibm,get-indicator-indices-types"** property in the **/rtas** node if the call is to be used for getting indicator information and must include the **"ibm,get-sensor-indices-types"** property in the **/rtas** node if the call is to be used for getting sensor information.

R1–7.4.0–15. When an indicator token is provided in the **"ibm,get-indicator-indices-types"** property, it must not be included in the **"**<*vendor*>**,indicator-**<*token*>**"** property and must not be included in the **"rtas-indicators"** property.

R1–7.4.0–16. When a sensor token is provided in the **"ibm,get-sensor-indices-types"** property, it must not be included in the **"**<*vendor*>**,sensor-**<*token*>**"** property and must not be included in the **"rtas-sensors"** property.

R1–7.4.0–17. When an environmental sensor token is provided in the **"ibm,get-sensor-indices-types"** property, users of data in the **"ibm,environmental-sensors"** property for that sensor token must not assume that the indices are contiguous for that sensor token (that is, any of the indices between 0 and the maxindex, inclusive, may be missing).

R1–7.4.0–18. When the value of any index returned is 0xFFFFFFFF, the OS must use the *ibm,get-dynamic-sensor-state* and *ibm,set-dynamic-indicator* RTAS functions for this sensor or indicator, using the location code to identify the sensor or indicator.

R1–7.4.0–19. The OS must not call *get-sensor-state* or *get-indicator* with an index value of 0xFFFFFFFF.

7.4.1 *ibm,set-dynamic-indicator* RTAS Call

This RTAS call behaves as the RTAS *set-indicator* call, except that the instance of the indicator is identified by a location code instead of a index.

R1–7.4.1–1. Platforms that implement any indicators that are identified by location code instead of index (see Requirement R1–7.4.0–18) must implement the *ibm,set-dynamic-indicator* RTAS function.

R1–7.4.1–2. The RTAS function *ibm,set-dynamic-indicator* must implement the argument call buffer defined by Table 106, “ibm,set-dynamic-indicator Argument Call Buffer,” on page 236.

Table 106. *ibm,set-dynamic-indicator* Argument Call Buffer

Parameter Type	*Name*	Values
In	*Token*	Token for *ibm,set-dynamic-indicator*
	Number Inputs	3
	Number Outputs	1
	Indicator	Token defining the indicator 9006: Error Log 9007: Identify Indicator
	State	Desired new state; see Table 40, “Defined Indicators,” on page 143.
	Location Code Address	Real or Logical address of a location code string, in the format defined by Requirement R1–7.4.1–5
Out	*Status*	-1: Hardware error -2: Busy, try again later -3: No such indicator 0: Success 990x: Extended delay, where x is a number between 0 and 5, as described below

When 990x *Status* is returned, it is suggested that software delay for 10 raised to the power *x* milliseconds (where *x* is the last digit of the 990x return code), before calling *ibm,set-dynamic-indicator* with the same indicator type and location code. However, software may call *ibm,set-dynamic-indicator* again either earlier or later than this.

R1–7.4.1–3. The OS must not call *ibm,set-dynamic-indicator* with a different indicator until a non-busy return *Status* has been received from the previous *ibm,set-dynamic-indicator* call.

R1–7.4.1–4. The location code string referenced by the *Location Code Address* argument in the *ibm,set-dynamic-indicator* argument call buffer must reside in contiguous in real memory below an address of 4GB.

R1–7.4.1–5. The input data format in the work area must be as follows:

a. 32-bit integer length of the location code string, including NULL

b. Location code string, NULL terminated, identifying the sensor to be set.

R1–7.4.1–6. The platform must not modify the location code string.

R1–7.4.1–7. The OS must only use this call for indicators which have been provided by the *ibm,get-indices* RTAS call with an index value of 0xFFFFFFFF.

R1–7.4.1–8. Platforms must identify all indicators except types 9006 and 9007 by index.

R1–7.4.1–9. The *ibm,set-dynamic-indicator* RTAS call must return A *Status* of -3 for the following conditions:

a. Indicator type not supported

b. The specified location code does not identify a valid indicator

7.4.2 *ibm,get-dynamic-sensor-state* RTAS Call

This RTAS call behaves as the RTAS *get-sensor-state* call, except that the instance of the sensor is identified by a location code instead of a index.

R1–7.4.2–1. Platforms that implement any sensors that are identified by location code instead of index (see Requirement R1–7.4.0–18) must implement the *ibm,get-dynamic-sensor-state* RTAS function.

R1–7.4.2–2. The RTAS function *ibm,get-dynamic-sensor-state* must implement the argument call buffer defined by Table 107, “ibm,get-dynamic-sensor-state Argument Call Buffer,” on page 237.

Table 107. *ibm,get-dynamic-sensor-state* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,get-dynamic-sensor-state*
	Number Inputs	2
	Number Outputs	2
	Sensor	Token defining the sensor
	Location Code Address	Real or Logical address of a location code string, in the format defined by Requirement R1–7.4.2–5
Out	*Status*	-1: Hardware error -2: Busy, try again later -3: No such indicator 0: Success 990x: Extended delay, where x is a number between 0 and 5, as described below
	State	Current state of the sensor as defined in the *Defined Values* column of Table 42, “Defined Sensors,” on page 147

When 990x *Status* is returned, it is suggested that software delay for 10 raised to the power *x* milliseconds (where *x* is the last digit of the 990x return code), before calling *ibm,get-dynamic-sensor-state* with the same indicator type and location code. However, software may call *ibm,get-dynamic-sensor-state* again either earlier or later than this.

R1–7.4.2–3. The OS must not call *ibm,get-dynamic-sensor-state* with a different sensor until a non-busy return *Status* has been received from the previous *ibm,get-dynamic-sensor-state* call.

R1–7.4.2–4. The work area must be contiguous in real memory and must reside below 4GB.

R1–7.4.2–5. The input data format in the work area must be as follows:

a. 32-bit integer length of the location code string, including NULL

b. Location code string, NULL terminated, identifying the sensor to be set.

R1–7.4.2–6. The platform must not modify the location code string.

R1–7.4.2–7. The OS must only use this call with sensors which have been provided by the *ibm,get-indices* RTAS call with an index value of 0xFFFFFFFF.

R1–7.4.2–8. The platform must use the *ibm,get-dynamic-sensor-state* RTAS call only for dynamic sensor types of 9004, 9006 and 9007.

R1–7.4.2–9. A *Status* of -3 must be returned for the following conditions:

a. Sensor type not supported

b. The specified location code does not identify a valid sensor

7.4.3 *ibm,get-vpd* RTAS Call

This RTAS call allows for collection of VPD that changes after OS boot time (after the initial reporting in the OF device tree). When this call is implemented, there is no overlap between what is reported in the device tree and what is reported with this RTAS call. Also, when this RTAS call is implemented, all VPD, except PCI and I/O device VPD, which is dynamically changed during OS run time is reported with this call and not via the **`"ibm,vpd"`** property in the OF device tree.

R1–7.4.3–1. For all Dynamic Reconfiguration options except PCI Hot Plug, when the platform VPD can change dynamically due to a Dynamic Reconfiguration operation, the platform must implement the *ibm,get-vpd* RTAS call.

R1–7.4.3–2. The RTAS function *ibm,get-vpd* must implement the argument call buffer defined by Table 108, “ibm,get-vpd Argument Call Buffer,” on page 239.

Table 108. *ibm,get-vpd* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,get-vpd*
	Number Inputs	4
	Number Outputs	3
	Pointer to Location Code	Real address of NULL-terminated string, contiguous in real memory and below 4GB, which is the location code of the FRU for which to obtain the VPD. When this parameter references a NULL string the VPD for all location codes is returned.
	Work Area Address	Address of work area
	Work Area Size	Size of work area
	Sequence Number	Integer representing the sequence number of the call. First call in sequence starts with 1, following calls (if necessary) use the *Next Sequence Number* returned from the previous call.
Out	*Status*	-1: Hardware error -3: Parameter error -4: Optional: VPD changed, start again 0: Success 1: More data available; call again 990x: Extended Delay where x is a number 0-5 (see text below)
	Next Sequence Number	Return this integer as the *Sequence Number* parameter on the next call to continue the sequence, or 1 if no more calls are required
	Bytes Returned	Integer representing the number of valid bytes returned in the work area.

When the 990x *Status* is returned, it is suggested that software delay for 10 raised to the x milliseconds (where x is the last digit of the 990x return code), before calling *ibm,get-vpd* with the same input parameters. However, software may issue the *ibm,get-vpd* call again either earlier or later than this.

R1–7.4.3–3. On the first call to *ibm,get-vpd* for a particular VPD gathering operation, the caller must provide a *Sequence Number* of 1 (32-bit integer)

R1–7.4.3–4. Upon calling *ibm,get-vpd* with a *Sequence Number* of 1, a previously returned *Next Sequence Number* must be discarded. This means that multiple calls to *ibm,get-vpd* cannot be interleaved by multiple processors, and if processor “B” starts a new *ibm,get-vpd* sequence while processor “A” has a call sequence in process (that is, the function on processor “A” has returned a *Status* of 1, and the subsequent call has not yet been made) then the call sequence on processor “A” is abandoned.

R1–7.4.3–5. Optionally, if firmware detects a change in the VPD being requested before the entire VPD is returned, the *ibm,get-vpd* call must return a *Status* of -4 and the caller must start again with a Starting Number of 1.

Implementation Note: The platform should not impede forward progress by continuously returning a *Status* of -4.

R1–7.4.3–6. The return data format in the work area must be such that after returning all the data and concatenating all data together in the order received, that the data is the same as is obtained from the **`“ibm,vpd”`** property of the OF device tree.

R1–7.4.3–7. Each stanza of the returned data must include the YL (location code) keyword.

R1–7.4.3–8. If the *ibm,get-vpd* RTAS call is implemented, then the platform must not provide the **"ibm,vpd"** or **"ibm,loc-code"** properties in the OF device tree **root** node.

R1–7.4.3–9. If the *ibm,get-vpd* RTAS call is implemented, then any VPD which may change after OS boot must be reported via the *ibm,get-vpd* RTAS call.

R1–7.4.3–10. If the *Status* returned is 1 (more data available, call again), then the caller must call *ibm,get-vpd* again with the *Sequence Number* parameter set to the *Next Sequence Number* integer from the previously returned call.

R1–7.4.3–11. If a *Status* of anything other than 0 or 1 is returned, the contents of the work area is not defined.

R1–7.4.3–12. The work area must be contiguous in real memory and must reside below 4GB.

R1–7.4.3–13. Firmware cannot count on the contents of the work area at the beginning of any call to *ibm,get-vpd* (regardless of the value of the *Sequence Number*).

R1–7.4.3–14. The location code referenced by the *Pointer to Location Code* parameter must reside in contiguous real memory below an address of 4GB.

R1–7.4.3–15. If the *ibm,get-vpd* RTAS call is implemented, then firmware must supply the **"ibm,vpd-size"** property in the **/rtas** node, the value of which is a single cell, encoded as with **encode-int**, which is the estimated maximum size in bytes of the VPD that is returned if the *Pointer to Location Code* parameter to the *ibm,get-vpd* RTAS function is NULL (that is, all system VPD). This size should take in to account possible concurrent addition of new platform elements after the partition is started. If firmware is unable to estimate this size, it may return a value of 0x0 to indicate that no estimate is available.

Software Implementation Notes:

1. An OS should be prepared for older versions of firmware where the **"ibm,vpd-size"** property is not provided.
2. Each stanza of the returned data must include the YL (location code) keyword.

7.4.4 Managing Storage Preservation

Platforms may optionally preserve selected regions of storage (LMBs) across client program boot cycles. See Section 2.1.3.6.12, "Persistent Memory and Memory Preservation Boot (Storage Preservation Option)," on page 49 for more information.

R1–7.4.4–1. For the Storage Preservation option: The platform must implement the *ibm,manage-storage-preservation* RTAS argument call buffer as defined by Table 109, "ibm,manage-storage-preservation Argument Call Buffer," on page 241.

Table 109. i*bm,manage-storage-preservation* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,manage-storage-preservation*
	Number Inputs	3
	Number Outputs	2
	Subfunction	0 = unused (return -3) 1 = Register specified LMB for preservation 2 = Query preservation state of specified LMB 3 = Deregister for preservation Specific LMB 4 = Deregister for preservation all caller's LMBs. All other values reserved (return -3)
	Reg High	The high order 32 bits of the LMB's **"reg"** property (Subfunctions 1-3)
	Reg Low	The low order 32 bits of the LMB's **"reg"** property (Subfunctions 1-3)
Out	*Status*	-1: Hardware error -2: Busy -3: Parameter error (Subfunction or Reg invalid; or Reg for a non-preservable LMB) 0: Success 990x: Extended delay where x is a number 0-5
	Preservation state	If *Status* = Success, the current preservation state of specified LMB (Subfunctions 1-3)

R1–7.4.4–2. For the Storage Preservation option: The platform must include the **"ibm,preservable"** property in the **/memory** nodes of its OF device tree, containing a value which reflects the platform's ability to preserve the specific LMB.

R1–7.4.4–3. For the Storage Preservation option: The value of the **"ibm,preservable"** property for the first LMB must be 0 (cannot be preserved).

R1–7.4.4–4. For the Storage Preservation option: The platform must not preserve the first LMB, thus must indicate a value of 0 for the **"ibm,preservable"** property for the first LMB.

R1–7.4.4–5. For the Storage Preservation option: The platform must include the **"ibm,preserved"** property in the **/memory** nodes of its OF device tree, valued to reflect the platform's preservation state of the specific LMB.

R1–7.4.4–6. For the Storage Preservation option: The platform, on a reboot, must include in the OF **/rtas** node the **"ibm,preserved-storage"** property if the previous client program registered one or more of its LMBs for preservation.

R1–7.4.4–7. For the Storage Preservation option: If the client program registered an LMB for preservation, the platform must preserve the LMB's storage state across client program reboots.

R1–7.4.4–8. For the Storage Preservation option: The platform, on a reboot, must include in the OF **/rtas** node the **"ibm,request-partition-shutdown"** property which reflects the value of the partition shutdown configuration variable, and if this property is not present, a value of 0 must be assumed by the OS.

7.4.5 *ibm,lpar-perftools* RTAS Call

This RTAS call provides access to platform-level facilities for performance tools running in a partition on an LPAR system. Platforms may require platform-specific tools, beyond the scope of this architecture, to make this call available.

R1–7.4.5–1. For the Performance Tool Support option: The platform must implement the LPAR option.

R1–7.4.5–2. For the Performance Tool Support option: RTAS must implement the *ibm,lpar-perftools* call using the argument call buffer defined by Figure 110, “ibm,lpar-perftools Argument Call Buffer,” on page 242.

Table 110. *ibm,lpar-perftools* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,lpar-perftools*
	Number Inputs	5
	Number Outputs	2
	Subfunction	1: Convert hypervisor IAR value to method name.
	Work Area Address High	Most significant 32 bits of real address of work area
	Work Area Address Low	Least significant 32 bits of real address of work area
	Work Area Size	Size of work area in bytes
	Sequence Number	Integer representing the sequence number of this call. First call in sequence starts with 1, following calls (if necessary) use the *Next Sequence Number* returned from the previous call.
Out	*Status*	-1: Hardware Error -2: Busy -3: Parameter Error (Subfunction invalid, invalid work area address, invalid work area size) -9002: Partition does not have authority to perform this function -5: Buffer was too small to supply requested data 0: Success 990x: Extended delay
	Next Sequence Number	Return this integer as the *Sequence Number* parameter on the next call, or 1 if no more calls are required.

When 990x *Status* is returned, it is suggested that software delay for 10 raised to the x milliseconds (where x is the last digit of the 990x return code), before calling the *ibm,lpar-perftools* call with the same input parameters. However, software may issue the *ibm,lpar-perftools* call again earlier or later than this.

R1–7.4.5–3. For the Performance Tool Support option: For *subfunction* value 1, on input the first 8 bytes of the work area must contain the hypervisor IAR address to be converted. On output, the first 8 bytes of the work area contain the offset of this address from the start of the hypervisor function, method or module, followed by a NULL-terminated text string containing the name of the hypervisor function, method or module. If the address is not a valid address in the hypervisor, on output the buffer must contain 0x0 (8 bytes) followed by a NULL-terminated text string indicating that the address was not valid.

R1–7.4.5–4. For the Performance Tool support option: The work area must reside in contiguous memory.

R1–7.4.5–5. For the Performance Tool Support option: If a *Status* of anything other than 0 is returned, the contents of the work area are not defined.

R1–7.4.5–6. For the Performance Tool Support option: A partition must have at most one call to this function in process at a given time. This means that if one processor in the partition initiates this call, receives a Busy or Extended Delay return, and then another processor calls this function with a sequence number of 1, a subsequent call using the *Next Sequence Number* returned to the first processor results in a Parameter Error return code.

7.4.6 *ibm,suspend-me* RTAS Call

The *ibm,suspend-me* RTAS call provides the calling OS the ability to suspend processing. Suspension of processing is required as part of OS hibernation or migration to another platform. This RTAS call is made by the last active processor thread of a partition. The OS uses the H_JOIN hcall() (see Section 14.11.5.1, “H_JOIN,” on page 466) to deactivate other processing threads. Processing treads may exit H_JOIN due to an unmaskable interrupt; if a thread has exited H_JOIN, *ibm,suspend-me* fails with a status of “multiple processor threads active”. The wake up from suspension is triggered by partition state change (see Section 17.8.7.1, “Partition Migration,” on page 738 and Section 17.8.7.2, “Partition Hibernation,” on page 740). The *ibm,suspend-me* RTAS call returns only on the calling virtual processor. Other virtual processors that were inactive when *ibm,suspend-me* was called remain so until they are proded by the OS.

While the logical configuration of a suspended partition remains static, the physical properties may change; the OS may wish to issue *ibm,update-nodes* (see *Section 7.4.7, “ibm,update-nodes RTAS Call,” on page 246*) to determine if any device tree nodes changed, and then refresh its view of the device tree physical properties using *ibm,update-properties* (see Section 7.4.8, “ibm,update-properties RTAS Call,” on page 249) and/or *ibm,configure-connector* (see Section 13.5.3.5, “ibm,configure-connector RTAS Call,” on page 369). Also during suspension, some system parameters may have changed. See Table 112, “System Parameters that May Change During Partition Migration and Hibernation,” on page 245, for details. The OS may want to re-scan selected system parameters.

R1–7.4.6–1. For the Partition Suspension option: The platform must implement the Logical Partitioning option (see Chapter 14, “Logical Partitioning Option,” on page 385).

R1–7.4.6–2. For the Partition Suspension option: RTAS must implement the *ibm,suspend-me* call within a logical partition using the argument call buffer defined by Table 111, “ibm,suspend-me Argument Call Buffer,” on page 243.

Table 111. *ibm,suspend-me* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,suspend-me*
	Number Inputs	0
	Number Outputs	1
Out	*Status*	9000: Suspension Aborted 0: Success -- expected return on function resume -1: Hardware Error -9004: Partition not suspendable -9005: Multiple processor threads active -9006: Outstanding COP Operations

R1–7.4.6–3. For the Partition Suspension option: The *ibm,suspend-me* RTAS call must determine that the calling partition is in the “suspendable state” (see Table 258, “VASI Migration Session States,” on page 739), else return a status of -9004 “Partition not suspendable”.

R1–7.4.6–4. For the Partition Suspension option: The *ibm,suspend-me* RTAS call must determine that the calling partition has no other active processor thread, else return a status of -9005 “Multiple processor threads active”.

R1–7.4.6–5. For the Partition Suspension option: The platform must implement the Thread Join option (see Section 14.11.5, “Thread Join Option,” on page 466).

R1–7.4.6–6. For the Partition Suspension option: The platform must restore all partition state as of the time of the call to *ibm,suspend-me* prior to returning from the *ibm,suspend-me* RTAS call, except for the values of those Open Firmware Device tree properties as reported using the Update OF Tree option, and the system parameters given in Table 112, “System Parameters that May Change During Partition Migration and Hibernation,” on page 245.

R1–7.4.6–7. For the Partition Suspension option: The platform must be prepared to respond to OS requests for updated device tree information immediately after returning from the *ibm,suspend-me* RTAS call.

R1–7.4.6–8. For the Partition Suspension option: The platform must support the “update OF tree” option.

R1–7.4.6–9. For the Partition Suspension option: The platform must support the “Partner partition suspended” CRQ Transport Event (See Table 232, “Transport Event Codes,” on page 622).

R1–7.4.6–10. For the Partition Suspension option: The *ibm,suspend-me* RTAS call must cause the platform to deliver “Partner partition suspended” CRQ Transport Events to both CRQs of all CRQ connections associated with the partition calling *ibm,suspend-me.*

Note: The transport events are visible to the partition calling *ibm,suspend-me* after the subsequent resume from suspension, while the transport events are immediately visible to the partner partitions of the caller at the time of the suspend.

R1–7.4.6–11. For the Partition Suspension option: The *ibm,suspend-me* RTAS call must cause the platform to set the state of all of the caller’s CRQs to disabled.

R1–7.4.6–12. For the Partition Suspension option: The platform must implement the H_ENABLE_CRQ hcall() using the syntax and semantics described in Section 17.2.3.1.5.4, “H_ENABLE_CRQ,” on page 641.

R1–7.4.6–13. For platforms that implement the Partition Suspension and VSCSI options: The **`“compatible”`** property of the platform’s **`v-scsi`** and **`v-scsi-host`** nodes must include `“IBM,v-scsi-2”` and `“IBM,v-scsi-host-2”` respectively indicating the platform supports the “Partner partition suspended” CRQ Transport Event.

R1–7.4.6–14. For the Partition Suspension option: If the OS is participating in OS surveillance, to avoid a surveillance time out, the OS must disable surveillance (see Section 7.3.5.2.1, “Surveillance,” on page 138) prior to calling *ibm,suspend-me*.

R1–7.4.6–15. For the Partition Suspension option: The platform must implement the LRDR option (See Section 13.7, “Logical Resource Dynamic Reconfiguration (LRDR),” on page 377).

R1–7.4.6–16. For the Partition Suspension option: The logical configuration of a partition, including its view of the *rtas-display-device*, and rtas tone facility must not change while a partition is suspended.

R1–7.4.6–17. For the Partition Suspension option: The platform must not change the support for a system parameter during suspension.

NOTE: If RTAS returns a status of -3 (System parameter not supported) prior to suspension, it returns a Status of -3 for accesses to that same system parameter after suspension. Similarly if RTAS does not return a Status of -3 prior to suspension for a given system parameter, it does not do so after suspension.

R1–7.4.6–18. For the Partition Suspension option: The platform must limit the system parameters that change values during suspension to those specified in Table 112, “System Parameters that May Change During Partition Migration and Hibernation,” on page 245 (all system parameters not specified are preserved).

R1–7.4.6–19. For the Partition Suspension option: The platform must preserve up to the first 32 SLB entries for each partition processor during the suspension. Other SLB entries are subject to loss.

R1–7.4.6–20. For the Partition Suspension option with the Platform Facilities Option: The *ibm,suspend-me* RTAS call must determine that the calling partition has no outstanding coprocessor operations else return a status of -9005 “Outstanding COP Operations”.

Table 112. System Parameters that May Change During Partition Migration and Hibernation

System Parameter Token	Name	
0-15	HMC	
18-19	Legacy processor CoD	
20	SPLPAR characteristics	Only specified SPLPAR keywords may change value
	DesiredEntitledCapacity	
	DesiredMemory	
	DesiredNumberOfProcessors	
	DesiredVariableCapacityWeight	
	DispatchWheelRotationPeriod	
	MinimumEntitledCapacityPerVP	Platform prevents migration where current Entitled Capacity/VCPU ratio would be below the target's minimum.
	MaximumPlatformProcessors	
22	platform_auto_power_restart	
23	sp-remote-pon	
24	sp-rb4-pon	
25	sp-snoop-str	
30	sp-call-home	
31	sp-current-flash-image	
33	epow3-quiesce-time	
34	memory-preservation-boot-time	
35	SCSI initiator identifier	
36	AIX support	The keyword “support” may not change to the value “no” for an AIX client.
37	enhanced processor CoD	

Table 112. System Parameters that May Change During Partition Migration and Hibernation *(Continued)*

System Parameter Token	Name	
38	enhanced memory CoD	
39	CoD Options	
41	firmware boot options	
43	processor module information	

7.4.7 *ibm,update-nodes* RTAS Call

This RTAS call is used to determine which device tree nodes have changed due to a massive platform reconfiguration such as when the partition is migrated between machines. Differing platform reconfigurations are expected to potentially result in different sets of nodes being updated; the “scope” argument communicates what set of changes are to be reported. The work area is a 4 KB naturally aligned area of storage below the first 4 GB; as such, it may not be large enough to contain the reports of all changed nodes. The status value of 1 is used to inform the caller that there are more updates to report and that it will have to call the *ibm,update-nodes* RTAS again to receive them. On subsequent calls the state variable, which is set to zero on the first call, is set to the value returned on the previous call, to supply RTAS with the information it needs to continue from where the previous call ended.

Upon return, the work area contains, in addition to the state variable, zero or more operation lists, and logically ends with a terminator (4 byte word naturally aligned containing 0x00000000). An operation list consists of an operator (4 bytes naturally aligned) and zero or more (up to a the maximum number of 4 byte locations remaining in the work area) operands, each 4 bytes long. An operator consists of a single byte opcode followed by 3 bytes encoded with the binary value of the number of operands that follow. An operator with an operand length field of zero performs no operation, and the opcode of zero is reserved for the terminator -- thus the terminator can be considered a special encoding of a no-op operator.

- The opcode of 0x01 is used for deleted nodes -- the operands are the **`phandle`** values for the deleted nodes.
- The opcode of 0x02 is used for updated nodes -- the operands are the **`phandle`** values for the updated nodes. The updated properties are obtained using the *ibm,update-properties* RTAS call.
- The opcode of 0x03 is used for adding nodes -- the operands are pairs of **`phandle`** and **`ibm,drc-index`** values; the **`phandle`** value denotes the parent node of the node to be added and the **`ibm,drc-index`** value is passed with the *ibm,configure-connector* RTAS call to obtain the contents of the added node.

To make processing of device tree updates simpler, all opcode 0x01 (delete) operations (if any) are presented prior to all opcode 0x02 (update) operations (if any), and finally any 0x03 (addition) operations are presented. The **`phandle`** operand values are the same **`phandle`** values as reported by the **`“ibm,phandle”`** property.

R1–7.4.7–1. For the Update OF Tree option: The platform must include the **`“ibm,phandle”`** property in all OF nodes specified in Table 117, “Nodes That May be Reported by ibm,update-nodes for a Given Value of the “Scope” Argument,” on page 248.

R1–7.4.7–2. For the Update OF Tree option: The platform must implement the *ibm,update-nodes* RTAS call using the argument call buffer defined by Table 113, “ibm,update-nodes Argument Call Buffer,” on page 247.

Table 113. *ibm,update-nodes* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,update-nodes*
	Number Inputs	2
	Number Outputs	1
	Work Area Address	32 bit real address of work area
	Scope	Values per Table 117, “Nodes That May be Reported by ibm,update-nodes for a Given Value of the “Scope” Argument,” on page 248.
Out	*Status*	-1: Hardware Error -2: Busy -3: Parameter Error (Purpose does not match the current partition state) 0: Success 1: More nodes updated -- call again

R1–7.4.7–3. The *ibm,update-nodes* RTAS call work area must be 4 KB long aligned on a 4 KB boundary that is accessible with MSR[DR] = 0, else RTAS may return -3 “Parameter Error”.

R1–7.4.7–4. The work area on the first call to *ibm,update-nodes* RTAS for a given value of “*Scope*” must be formatted as specified in Table 114, “Initial Format of Work Area for ibm,update-nodes,” on page 247, else RTAS may return -3 “Parameter Error”.

Table 114. Initial Format of Work Area for *ibm,update-nodes*

0x00000000 (State Variable indicates Initial call for specified *Scope*)
12 bytes of 0x00 (reserved)
Don’t Care . . .

R1–7.4.7–5. Upon successful return (non-negative status value) from *ibm,update-nodes* the work area must by formatted as defined in Table 115, “Format of Work Area for ibm,update-nodes,” on page 247. (Note each entry in Table 115 is 4 bytes long.)

Table 115. Format of Work Area for *ibm,update-nodes*

State Variable (4 Bytes)
12 bytes of 0x00 (reserved)
0 or more operation lists
. . .
Terminator (0x00000000)

R1–7.4.7–6. An *ibm,update-nodes* RTAS call operation list for the *ibm,update-nodes* RTAS call must contain an operator (4 bytes naturally aligned) and zero or more 4 byte operands up to the end of the work area.

R1–7.4.7–7. An operator in an *ibm,update-nodes* RTAS call operation list must be formatted with, starting at the high order byte, a single byte opcode followed by 3 bytes encoded with the binary value of the number of operands that follow.

R1–7.4.7–8. An operator in an *ibm,update-nodes* RTAS call operation list with an operand length field of zero must be considered to perform no operation.

R1–7.4.7–9. The opcode of 0x01 in an *ibm,update-nodes* RTAS call operation list must be used to denote node deletions.

R1–7.4.7–10. The operands for opcode 0x01 in an *ibm,update-nodes* RTAS call operation list must be the **`phandle`** values for the deleted nodes.

R1–7.4.7–11. The opcode of 0x02 in an *ibm,update-nodes* RTAS call operation list must be used to denote updated nodes.

R1–7.4.7–12. The operands for opcode 0x02 in an *ibm,update-nodes* RTAS call operation list must be the **`phandle`** values for the updated nodes that may be used as the *ibm,update-properties* RTAS call argument to obtain the changed properties of the updated node.

R1–7.4.7–13. The opcode of 0x03 in an *ibm,update-nodes* RTAS call operation list must used for the added nodes.

R1–7.4.7–14. The operands for opcode of 0x03 in an *ibm,update-nodes* RTAS call operation list must be **`phandle`** and **`ibm,drc-index`** value pairs (each value being 4 bytes on a natural boundary totalling 8 bytes for the pair) denoting the parent node of the added node and the *ibm,configure-connector* RTAS call argument to obtain the contents of the added node respectively.

R1–7.4.7–15. All opcode 0x01 (delete) in an *ibm,update-nodes* RTAS call operation list (if any) must be presented prior to any opcode 0x02 (update) operations (if any).

R1–7.4.7–16. All opcode 0x02 (update) in an *ibm,update-nodes* RTAS call operation list (if any) must be presented prior to any opcode 0x03 (add) operations (if any).

R1–7.4.7–17. The work area on subsequent call(s) to *ibm,update-nodes* RTAS for the same value of the "*Scope"* must be formatted as specified in Table 116, "Format of Work Area for Subsequent Calls to ibm,update-nodes," on page 248, else RTAS may return -3 "Parameter Error".

Table 116. Format of Work Area for Subsequent Calls to *ibm,update-nodes*

Value of the 1st 16 bytes of the returned work area from last call to *ibm,update-nodes* RTAS that returned status of 1.
Don't Care . . .

R1–7.4.7–18. The "*Scope*" argument for the *ibm,update-nodes* RTAS call must be one of the values specified in the scope value column of Table 117, "Nodes That May be Reported by ibm,update-nodes for a Given Value of the "Scope" Argument," on page 248, else RTAS may return -3 "Parameter Error".

R1–7.4.7–19. For the *ibm,update-nodes* RTAS call, the platform must restrict its reported node updates to those specified in Table 117, "Nodes That May be Reported by ibm,update-nodes for a Given Value of the "Scope" Argument," on page 248 for the value of the specified *"Scope"* argument.

Table 117. Nodes That May be Reported by *ibm,update-nodes* for a Given Value of the "*Scope*" Argument

Scope Value	Reportable node types (value of **`"name"`** or **`"device_type"`** property)	Supported Opcodes

Table 117. Nodes That May be Reported by *ibm,update-nodes* for a Given Value of the "*Scope*" Argument

Negative values: Platform Resource Reassignment events as from *event-scan* RTAS	cpu	**0x02**
	memory	**0x02**
	ibm,dynamic-reconfiguration-memory	**0x02**
	ibm,plaform-facilities	0x01-0x03
	ibm,random-v#	0x01-0x03
	ibm,compression-v#	0x01-0x03
	ibm,encryption-v#	0x01-0x03
1 Partition Migration / Hibernation	**root**	**0x02**
	openprom	**0x02**
	rtas	**0x02**
	vdevice	**0x02**
	cpu	**0x02**
	cache	**0x01-0x03**
	options	**0x02**
	memory	**0x02**
	ibm,dynamic-reconfiguration-memory	<all>
	ibm,platform-facilities	0x01-0x03
	ibm,random-v#	0x01-0x03
	ibm,compression-v#	0x01-0x03
	ibm,encryption-v#	0x01-0x03
2 Activate Firmware	**rtas**	0x02

7.4.8 *ibm,update-properties* RTAS Call

This RTAS call is used to report updates to the properties changed due to a massive platform reconfiguration such as when the partition is migrated between machines. This RTAS call reports changes in the node specified by the phandle value in the work area passed using the Work Area Address argument. The phandle value may be that of a critical node that the caller is interested in or one reported by *ibm,update-nodes* RTAS call. These changes may include any combination of new values, deleted and added properties. Updates for a given node are retained until the platform is subsequently reconfigured, and remain available to subsequent calls to *ibm,update-nodes*.

There may be more changes than can be reported in a single 4 K work area. In this case, the RTAS call returns with a status of 1 "More properties updated -- call again". On the first call, the second word of the work area contains the value 0 specifying that the RTAS call is to start with the first changed property for the specified updated node. On a call with a status value of 1, the first sixteen (16) bytes of the work area contain values that, when subsequently supplied in the work area of another call to *ibm,update-properties* RTAS, specify that the call returns the updated property data for properties after those reported in the previous call.

A single updated property value string may exceed the capacity of a single 4 K work area. In that case, the updated property value descriptor for the property appears in the work area of multiple sequential calls to *ibm,update-proper-*

ties RTAS. When the updated property value descriptor contains the final data for the property value, the property string length field of the updated property value descriptor is a positive number. When the updated property value descriptor contains either the initial or interim data for the property value, the updated property string length field is a negative number denoting the twos complement of the length of the updated property string contained in the work area. The data value strings for a given property name are concatenated until the final updated property value descriptor is processed.

The first value returned, with an updated property name string of NULL, is always the node's name (for example: full path || **`name`** property value || @ unit address) even if there has been no change.

R1–7.4.8–1. For the Update OF Tree option: The platform must implement the *ibm,update-properties* RTAS call using the argument call buffer defined by Table 118, "ibm,update-properties Argument Call Buffer," on page 250.

Table 118. *ibm,update-properties* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,update-properties*
	Number Inputs	2
	Number Outputs	1
	Work Area Address	32 bit real address of work area
	Scope	Values per Table 122, "Properties of the Nodes That May Be Reported by ibm,update-properties for a "Scope"," on page 252.
Out	*Status*	-1: Hardware Error -2: Busy -3: Parameter Error (Purpose does not match the current partition state) 0: Success 1: More properties updated -- call again

R1–7.4.8–2. The work area for the *ibm,update-properties* RTAS call must be 4 KB long aligned on a 4 KB boundary that is accessible with MSR[DR] = 0, else RTAS may return -3 "Parameter Error".

R1–7.4.8–3. The work area on the first call to *ibm,update-properties* RTAS for a given updated node must be formatted as specified in Table 119, "Initial Format of Work Area for ibm,update-properties," on page 250, else RTAS may return -3 "Parameter Error".

Table 119. Initial Format of Work Area for *ibm,update-properties*

`phandle` of updated node containing updated properties to be reported (4 bytes)
0x00000000 (Indicates Initial call for specified **`phandle`**)
8 bytes of 0x00 (reserved)
Don't Care . . .

R1–7.4.8–4. Upon successful return (non-negative status value) from *ibm,update-properties* the work area must by formatted as defined in Table 120, "Return Format of Work Area for ibm,update-properties," on page 251.

Table 120. Return Format of Work Area for *ibm,update-properties*

Description	Comments
phandle of updated node containing updated properties to be reported.	4 Bytes
State Variable (to be returned if status argument value = 1)	4 Bytes
Reserved	8 bytes
Number of properties reported in the work area	4 Bytes The number (N) of updated property value descriptors that follow -- see below
N updated property value descriptors	Each updated property value descriptor is formatted as: Null terminated string indicating the name of the updated property. followed by: Value Descriptor -- 4 Bytes decoded as 0x00000000 Name only property (**"encode-null"**) no value follows 0x80000000 The property is to be deleted no value follows Other positive values are the length (M) of the immediately following property value string that completes the update of the property value. Other negative values are the twos complement of the length (M) of the immediately following property value string that either starts or continues the update of the property value with the remainder in the work area of subsequent call(s) to *ibm,update-properties*. Followed by: 0-M bytes of property value string.

R1–7.4.8–5. Upon successful return (non-negative status value) from *ibm,update-properties* when the State Variable had been 0x00000000, the first updated property descriptor must describe the fully qualified path name of the node specified by the phandle argument in the work buffer; the three fields of this updated property descriptor are:

- Property name string is as from **encode-null**
- Value descriptor is the 4 byte binary length of the value string
- Value string is the fully qualified path name as from the node name string returned by the open firmware **package-to-path** client interface call.

R1–7.4.8–6. The work area on subsequent call(s) to *ibm,update-properties* RTAS for a given updated node must be formatted as specified in Table 121, "Format of Work Area for Subsequent Calls to ibm,update-properties," on page 251, else RTAS may return -3 "Parameter Error".

Table 121. Format of Work Area for Subsequent Calls to *ibm,update-properties*

phandle of updated node containing updated properties to be reported (4 Bytes)
Value from last call to *ibm,update-properties* RTAS that returned status of 1 (12 bytes).
Don't Care . . .

R1–7.4.8–7. For the *ibm,update-properties* RTAS call, the platform must restrict its reported property updates to those specified in Table 122, “Properties of the Nodes That May Be Reported by ibm,update-properties for a “Scope”,” on page 252 for the value of the specified *“Scope”* argument.

R1–7.4.8–8. For the *ibm,update-properties* RTAS call, the platform must return a *Status* of -3 (Parameter Error) for an unrecognized value of the “Scope” argument.

Table 122. Properties of the Nodes That May Be Reported by *ibm,update-properties* for a “*Scope*”

Scope Value	Reportable node types (value of **`"name"`** or **`"device_type"`** property)	Property Name
All negative values: Resource Reassignment events as from event-scan RTAS	/memory	“ibm,associativity”
	ibm,dynamic-reconfiguration-memory	“ibm,dynamic-memory”
	cpu	“ibm,associativity”
	`ibm,random-v#`	<all>
	`ibm,compression-v#`	<all>
	`ibm,encryption-v#`	<all>

Table 122. Properties of the Nodes That May Be Reported by *ibm,update-properties* for a "*Scope*" ***(Continued)***

1 Partition Migration / Hibernation	**root**	**"ibm,model-class"**
		"clock-frequency"
		"ibm,extended-clock-frequency"
		"model"
		"compatible"
		"name"
		"system-id"
		"ibm,partition-no"
		"ibm,drc-indexes"
		"ibm,drc-names"
		"ibm,drc-power-domains"
		"ibm,drc-types"
		"ibm,aix-diagnostics"
		"ibm,diagnostic-lic"
		"ibm,platform-hardware-notification"
		"ibm,ignore-hp-po-fails-for-dlpar"
		"ibm,managed-address-types"
		"ibm,service-indicator-mode"
	openprom	**"model"**
	rtas	**"power-on-max-latency"**
		"ibm,associativity-reference-points"
		"ibm,max-associativity-domains"
		"ibm,configure-kernel-dump"
		"ibm,configure-kernel-dump-sizes"
		"ibm,configure-kernel-dump-version"
		"ibm,read-slot-reset-state-functions"
		"ibm,configure-pe"
		"ibm,change-msix-capable"
	vdevice	**"ibm,drc-names"**
	children of the **vdevice** node	**"ibm,loc-code"**

Table 122. Properties of the Nodes That May Be Reported by *ibm,update-properties* for a "*Scope*" ***(Continued)***

1 Partition Migration / Hibernation	cpu	"name"
		"d-cache-sets"
		"d-cache-size"
		"i-cache-sets"
		"i-cache-size"
		"bus-frequency"
		"ibm,extended-bus-frequency"
		"ibm,extended-clock-frequency"
		"clock-frequency"
		"timebase-frequency"
		"l2-cache"
		"performance-monitor"
		"ibm,associativity"
		TLB properties (See Section C.6.1.5, "TLB properties," on page 772)
		"slb-size"
		"ibm,tbu40-offset"
		"ibm,pi-features"
		"ibm,spurr"
		"ibm,pa-optimizations"
		"ibm,dfp" (sign bit only)
		"ibm,sub-processors"
	cache	"d-cache-sets"
		"d-cache-size"
		"i-cache-sets"
		"i-cache-size"
		l2-cache
	options	"ibm,dasd-spin-interval"
	memory	"ibm,associativity"
	ibm,dynamic-reconfiguration-memory	"ibm,associativity-lookup-arrays"
		"ibm,dynamic-memory" only the associativity list index fields
		"ibm,memory-preservation-time"
	/chosen	"ibm,architecture-vec-5" byte 3 (I/O Super Page Option support parameters)

Table 122. Properties of the Nodes That May Be Reported by *ibm,update-properties* for a "*Scope*" ***(Continued)***

1 Partition Migration / Hibernation	**ibm,random-v#**	<all>
	ibm,compression-v#	<all>
	ibm,encryption-v#	<all>
2 Activate Firmware	**rtas**	1. Any **/rtas** node property as defined per LoPAPR remains invariant. 2. Any **/rtas** node property or definition extension, except for the value of a function token property*, may change (provided that the client program has indicated support for such property or definition extension) including the following: **"ibm,read-slot-reset-state-functions"** **"ibm,configure-pe"** *NOTE: This exception mandates that if an RTAS function token property survives a firmware activation, the token value of that RTAS function call does not change.

7.4.9 *ibm,configure-kernel-dump* RTAS call

This RTAS call is used to register and unregister with the platform a data structure describing kernel dump information. This dump information is preserved as needed by the platform in support of a platform assisted kernel dump option.

R1–7.4.9–1. For the Configure Platform Assisted Kernel Dump option: The platform must implement the *ibm,configure-kernel-dump* RTAS call using the argument call buffer defined by Table 123, "ibm,configure-kernel-dump Argument Call Buffer," on page 255.

Table 123. *ibm,configure-kernel-dump* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,configure-kernel-dump*
	Number Inputs	3
	Number Outputs	1
	Command	1: Register for future kernel dump 2: Unregister for future kernel dump 3: Complete/Invalidate current kernel dump
	Work_buffer_address	When command is 1: Register for future kernel dump, points to a structure as defined in Table 124, "Kernel Assisted Dump Memory Structure," on page 256
	Work_buffer_length	Length of Kernel Dump Memory Structure when defined above
Out	*Status*	0: Success -1: Hardware Error -2: Busy -3: Parameter Error -9: Dump Already Registered -10: Dump Active 990x:Extended Delay

R1–7.4.9–2. For the Configure Platform Assisted Kernel Dump option: The work-buffer address and work-buffer-length for the *ibm,configure-kernel-dump* RTAS call must point to an RMR-memory buffer that contains the structures described in Table 124, "Kernel Assisted Dump Memory Structure," on page 256, whenever the command is 1, register for future kernel dump; otherwise the call may return -3, "Parameter Error."

The Dump Memory Structure specified in Table 124, “Kernel Assisted Dump Memory Structure,” on page 256 is passed by the operating system during a *ibm,configure-kernel-dump* RTAS call. It is also reported by the platform using the *ibm,kernel-dump* RTAS property after a dump has been initiated.

Table 124. Kernel Assisted Dump Memory Structure

Header		
Offset	Number of Bytes	Value
0x0	4	Dump Format Version = 0x00000001
0x4	2	Number of Kernel Dump Sections
0x6	2	Dump Status Flags A bit mask with value 0x8000 = Dump performed (Set to 0 by caller of the *ibm,configure-kernel-dump* call) 0x4000 = Dump was triggered by the previous system boot (set by platform) 0x2000 = Dump error occurred (set by platform) All other bits reserved
0x8	4	Offset to first Kernel Dump Section, offset from the beginning of the Structure
0xc	4	Number of bytes in a block of the dump-disk, if data to be written to a dump-disk, If not, should be set to 0 (indicating the no disk dump option.)
0x10	8	Starting block# offset on dump-disk (set to 0 for the no disk dump option)
0x18	8	Number of blocks on dump-disk usable for dump (set to 0 for the no disk dump option)
0x20	4	Offset from start of structure to a Null-terminated Dump-disk path string (set to 0 for the no disk dump option)
0x24	4	Maximum time allowed (milliseconds) after Non-Maskable-Interrupt for the OS to call *ibm,configure-kernel-dump Function* 2 (unregister) to prevent an automatic dump-reboot (set to 0 to disable the automatic dump-reboot function)
Dump-disk Path String		
Offset specified above	Varies	Null-terminated Dump-disk path string specifying the dump-disk. If no disk dump option is indicated, this section is not included.
First Kernel Dump Section		
Offset specified above	4	Dump Request Flags: A bit-mask Bit 0x00000001 When set, firmware to copy source data to partition memory. This option must be selected if no disk dump option is indicated. All other bit values reserved
Section Start+4	2	Source Data type, describes section of dump memory being described 0x0001 = CPU State Data 0x0002 = Hardware Page Table for Real Mode Region 0x0011 = Real Mode Region 0x0012 = Dump OS identified string (identifies that the dump is for a particular OS type and version) 0x0100 - 0xFFFF OS defined source types All Other values reserved
Section Start+6	2	Dump Error Flags (set by platform) Bit mask 0x8000 = Invalid section data type 0x4000 = Invalid source address 0x2000 = Requested section length exceeds source 0x1000 = Invalid partition destination address 0x0800 = Partition memory destination too small

Table 124. Kernel Assisted Dump Memory Structure ***(Continued)***

Section Start+8	8	Source address (logical address if section came from partition memory, or byte offset if section is platform memory)
Section Start+16	8	Requested data length, represents number of bytes to dump
Section Start+24	8	Actual data length, number of bytes dumped
Section Start+32	8	Destination address, logical address used for sections not written to disk by the platform, always required for a Real mode region section and for all sections when the no disk dump option is used.
Subsequent Sections		
Previous Section Start+40	Start of Next Section	A total of up to nine additional 40 bytes sections with values as described in the First Section may be specified so long as the entire structure does not exceed 512 bytes for version 1.

R1–7.4.9–3. When the platform receives an *ibm,os-term* RTAS call, or on a system reset without an *ibm,nmi-interlock* RTAS call, if the platform has a dump structure registered through the *ibm,configure-kernel-dump* call, the platform must process each registered kernel dump section as required and, when available, present the dump structure information to the operating system through the **`"ibm,kernel-dump"`** property, updated with status for each dump section, until the dump has been invalidated through the *ibm,configure-kernel-dump* RTAS call.

R1–7.4.9–4. If *ibm.comfigure-kernel-dump* RTAS call is made to register or unregister for a dump while a dump is currently active, the platform must return a *Status* of -9, “Dump Active” indicating that a dump has been copied by the platform and a call must be made to complete/invalidate the active dump before another call to register or unregister a dump can be completed successfully.

R1–7.4.9–5. If *ibm.comfigure-kernel-dump* RTAS call is made to register a dump after a dump has already been registered by a call, the platform must return a *Status* of -8, “Dump Already Registered” unless an intervening call was made to invalidate the previously registered dump.

R1–7.4.9–6. For the Configure Platform Assisted Kernel Dump Option: Partition memory not specifically mentioned in the call structure must be preserved by the platform at the same location as existed prior to the os termination or platform reboot.

R1–7.4.9–7. For the Configure Platform Assisted Kernel Dump Option: The platform must present the RTAS property, **`"ibm,configure-kernel-dump-sizes"`** in the OF device tree, which describes how much space is required to store dump data for the firmware provided dump sections, where the firmware defined dump sections are:

- 0x0001 = CPU State Data
- 0x0002 = Hardware Page Table for Real Mode Region

R1–7.4.9–8. For the Configure Platform Assisted Kernel Dump Option: The platform must present the RTAS property, **`"ibm-configure-kernel-dump-version"`** in the OF device tree.

R1–7.4.9–9. For the Configure Platform Assisted Kernel Dump Option: After a dump registration is disabled (for example, by a partition migration operation), calls to *ibm,os-term* must return to the OS as though a dump was not registered.

Programming Note: The intended flow of interactions that utilize this call is as follows:

1. The OS registers sections of memory for dump preservation during OS initialization

2. The OS terminates abnormally
3. The Platform moves registered sections of memory as instructed during dump registration.
4. The Partition reboots and provides the prior registration data in the device tree.
5. The OS writes the preserved memory regions to disk before using those memory regions for regular use
6. The OS completes/invalidates current dump status.

7.4.10 DMA Window Manipulation Calls

DMA windows for a PE can be changed by the OS when the platform implements the Dynamic DMA Windows (DDW) option for a PE. The occurrence of the **"ibm,ddw-applicable"** property in any node of the OF Device Tree indicates that the platform implements the DDW option, but that property is required to be in the bridge above a PE in order for the DDW RTAS call to be applicable for the PE. That is, DDW may be applicable to some PEs in a platform and not for others.

The platform may implement the DDW RTAS calls even when the OS does not support these, because they are not required to be used by the OS, because there is always a default window initially allocated below 4 GB, as specified by the **"ibm,dma-window"** property. During partition migration, these RTAS calls may come and go, but so will the **"ibm,ddw-applicable"** property as the nodes in which those are supported come or go.

The following is an example of how an OS may grab all DMA window resources allocated for a PE:

1. If the default window (as specified by the **"ibm,dma-window"** property for the PE) is not needed, then call *ibm,remove-pe-dma-window* for the PE, specifying the default window LIOBN, to make the maximum resources available for the *ibm,create-pe-dma-window* RTAS call.

2. Call *ibm,query-pe-dma-window* for the PE to get the *Windows Available* and *PE TCEs* available for the PE. If the *Windows Available* field indicates 1 or more and the *PE TCEs* field is non-zero, then continue.

3. Call *ibm,create-pe-dma-window* for the PE, specifying the size based on the *PE TCEs* field obtained from the *ibm,query-pe-dma-window* RTAS call in step 2 and on the I/O page size being specified.

4. If the *Windows Available* field indicated 2 or more in step 2, then go back to step 2 and repeat, otherwise finished.

Software Implementation Note: The general expectation is that if the **"ibm,ddw-applicable"** property exists for a PE, that the OS will be able to generate one or more windows whose total size is larger than what is available via the default window. This requires either additional TCEs being available or that I/O page sizes other than 4 KB are available and the PE can use the largest I/O page size (the default window using only the 4 KB I/O page size). If not, then removing the default window would only allow re-allocation of the same size window at a different bus address (that is, same number of TCEs and same I/O page size). However, it may be possible for this to happen, in which case the platform may indicate that DDW is available to a PE, but removal of the default window will only allow creation of the same size window. An example is when a larger I/O page size is available but only the TCEs in the default window are available, and the PE cannot make use of the larger page size.

R1–7.4.10–1. For the Dynamic DMA Windows (DDW) option: The platform must implement all of the following RTAS calls: *ibm,query-dma-window*, *ibm,create-dma-window*, and *ibm,remove-dma-window*.

R1–7.4.10–2. For the Dynamic DMA Windows (DDW) option: The platform must provide the **"ibm,ddw-applicable"** property in the OF Device Tree in the bridge above each PE for which the DDW option is supported.

R1–7.4.10–3. For the Dynamic DMA Windows (DDW) option: The software must not call the *ibm,query-dma-window*, *ibm,create-dma-window*, or *ibm,remove-dma-window* RTAS calls in the absence of the **"ibm,ddw-applicable"** property for the PE, otherwise the call returns a *Status* of -3 (Parameter error), and when the property does exist, software must use the token values specified in the **"ibm,ddw-applicable"** property for these RTAS calls.

R1–7.4.10–4. For the Dynamic DMA Windows (DDW) option: The platform must provide a default DMA window for each PE, and all of the following must be true:

a. The window is defined by the **"ibm,dma-window"** property in the OF device tree.

b. The window is defined with 4 KB I/O pages.

c. The window is located entirely below 4 GB.

R1–7.4.10–5. For the Dynamic DMA Windows (DDW) option: The platform must remove any DMA windows created by the *ibm,create-pe-dma-window* RTAS call for a PE and must restore the default DMA window (if it was removed) for the PE, as originally defined by the **"ibm,dma-window"** properties for the PE, in each of the following cases:

a. On a reboot of the partition

b. On a DR isolate operation that encompasses the PE

R1–7.4.10–6. For the Dynamic DMA Windows (DDW) option: In Requirement R1–7.4.10–5, the platform must provide the same LIOBN, location, and size as specified in the **"ibm,dma-window"** property in the OF Device Tree for the device.

7.4.10.1 ibm,query-pe-dma-window

This RTAS call allows for the discovery of the resources necessary to make a successful subsequent call to *ibm,create-dma-window.*

R1–7.4.10.1–1. RTAS must implement a *ibm,query-pe-dma-window* call using the argument call buffer defined by Table 125, "ibm,query-pe-dma-window Argument Call Buffer," on page 260.

Table 125. *ibm,query-pe-dma-window* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,query-pe-dma-window*
	Number Inputs	3
	Number Outputs	5
	Config_addr	PE configuration address (Register fields set to 0)
	PHB_Unit_ID_Hi	Represents the most-significant 32-bits of the Unit ID of the PHB that corresponds to the *config_addr*
	PHB_Unit_ID_Low	Represents the least-significant 32-bits of the Unit ID of the PHB that corresponds to the *config_addr*

Table 125. *ibm,query-pe-dma-window* Argument Call Buffer ***(Continued)***

Parameter Type	Name	Values
Out	*Status*	0: Success -1: Hardware Error -3: Parameter error
	Windows Available	Number of additional DMA windows that can be created for this PE. If the value is 0 and the default window (as specified by the **"ibm,dma-window"** property for the PE) has not been yet removed via the *ibm,remove-pe-dma-window* RTAS call for that window, and if the default window is not needed, then removal of the default window makes at least one window available.
	PE TCEs	Largest contiguous block of TCEs allocated specifically for (that is, are reserved for) this PE. See also Requirement R1–7.4.10.1–2.
	IO Page Sizes	I/O Page Size Support. I/O page sizes supported for this PE. This is a bit significant field, defined as follows: Bits 0 - 23 reserved 24 = 16 GB page size supported 25 = 256 MB page size supported 26 = 128 MB pages supported 27 = 64 MB page size supported 28 = 32 MB page size supported 29 = 16 MB page size supported 30 = 64 KB page size supported 31 = 4 KB page size supported
	Migration Capable	H_MIGRATE_DMA Mask. Mask to indicate for which page sizes (as specified in the I/O Page Size Support field), that H_MIGRATE_DMA is supported (for this PE). This is a bit significant field, with the bits defined to align to the bits in the I/O Page Size Support field.

R1–7.4.10.1–2. For the Dynamic DMA Windows (DDW) option: TCE resources returned in the *PE TCEs* parameter of the *ibm,query-dma-window* RTAS call must be allocated to the PE specified by the PE configuration address specified in the call, and must be available to a subsequent *ibm,create-dma-window* RTAS call for that PE.

7.4.10.2 *ibm,create-pe-dma-window*

This call allows the creation of a new DMA window, given the size of the DMA window, I/O page size, and the PE to which it is associated. The return from the call includes the LIOBN of the new DMA window, the starting I/O address of the DMA window, and size.

Software Implementation Note: Software is expected to not attempt to create a DMA window that is larger than possible, or create more DMA windows than is possible, otherwise the *ibm,create-pe-dma-window* will return a *Status* of -3 (Parameter Error). Thus, the OS is expected to use the *ibm,query-pe-dma-window* first and not ask to create a window that consumes more resources than those that are available to the PE.

R1–7.4.10.2–1. For the Dynamic DMA Windows (DDW) option: RTAS must implement the *ibm,create-dma-window* call using the argument call buffer defined by Table 126, "ibm,create-pe-dma-window Argument Call Buffer," on page 262.

Table 126. *ibm,create-pe-dma-window* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,create-pe-dma-window*
	Number Inputs	5
	Number Outputs	4
	Config_addr	PE configuration address (Register fields set to 0)
	PHB_Unit_ID_Hi	Represents the most-significant 32-bits of the Unit ID of the PHB that corresponds to the *config_addr*
	PHB_Unit_ID_Low	Represents the least-significant 32-bits of the Unit ID of the PHB that corresponds to the *config_addr*
	I/O Page Size	The value n, where 2^n is the requested I/O page size. Only page sizes obtained from the *ibm,query-pe-dma-window* RTAS call for the PE are allowed. Values of n from 0-11 are invalid.
	Requested Window Size	The value n, where 2^n is the requested DMA window size.
Out	*Status*	990x: Extended delay, where x is a number 0-5 (see text) 0: Success (window created) -1: Hardware Error -2: Busy, try again later -3: Parameter Error
	LIOBN	LIOBN of the DMA window created by this call, if any. If no DMA window was created (that is, if the *Status* is not 0), then this field is present but not used.
	I/O Starting Address Hi	Represents the most-significant 32-bits of the starting address on the I/O bus for the DMA window created by this call, if any. If no DMA window was created (that is, if the *Status* is not 0), then this field is present but not used.
	I/O Starting Address Low	Represents the least-significant 32-bits of the starting address on the I/O bus for the DMA window created by this call, if any. If no DMA window was created (that is, if the *Status* is not 0), then this field is present but not used.

R1–7.4.10.2–2. For the Dynamic DMA Windows (DDW) option: The *ibm,create-pe-dma-window* RTAS call must return a *Status* of 0 (Success) only if a window with the requested attributes is created, and must not create a new window if a non-0 *Status* is returned.

7.4.10.3 *ibm,remove-pe-dma-window*

This RTAS call allows for the removal of PE DMA windows, including those created with the *ibm,create-pe-dma-window* RTAS call as well as the default window specified by the **`"ibm,dma-window"`** property for the PE. All created DMA windows will be removed by the platform, and the default DMA window restored, on a partition reboot, on a DR isolate operation (see Requirement R1–7.4.10–5 and R1–7.4.10–6), or if the last remaining DMA window for the PE is removed and that window is not the default DMA window (see Requirements R1–7.4.10.3–3 and R1–7.4.10.3–4). After removal of a DMA window, software needs to use the *ibm,query-pe-dma-window* RTAS call to find out what resources are available to the PE for subsequent *ibm,create-pe-dma-window* RTAS call.

R1–7.4.10.3–1. For the Dynamic DMA Windows (DDW) option: RTAS must implement the *ibm,remove-dma-window* call using the argument call buffer defined by Table 127, “ibm,remove-pe-dma-window Argument Call Buffer,” on page 263.

Table 127. *ibm,remove-pe-dma-window* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,remove-pe-dma-window*
	Number Inputs	1
	Number Outputs	1
	LIOBN	LIOBN of the DMA window to be removed.
Out	*Status*	0: Success -1: Hardware Error -3: Parameter error

R1–7.4.10.3–2. For the Dynamic DMA Windows (DDW) option: The caller of the *ibm,remove-pe-dma-window* RTAS call must assure that the TCE table specified by the *LIOBN* field does not contain any valid mappings at the time of the call (that is, that the window is not being used).

R1–7.4.10.3–3. For the Dynamic DMA Windows (DDW) option: The platform must restore the default DMA window for the PE on a call to the *ibm,remove-pe-dma-window* RTAS call when all of the following are true:

a. The call removes the last DMA window remaining for the PE.

b. The DMA window being removed is not the default window.

R1–7.4.10.3–4. For the Dynamic DMA Windows (DDW) option: In Requirement R1–7.4.10.3–3, the platform must provide the same LIOBN, location, and size as specified in the **`"ibm,dma-window"`** property in the OF Device Tree for the PE.

7.4.10.4 Extensions to Dynamic DMA Windows

Platforms supporting the DDW option implement extensions described in this section. These extensions include: adding the **`"ibm,ddw-extensions"`** property see Section B.6.5.1.1.1, “Properties for Children of PCI Host Bridges,” on page 703 to those nodes that include the **`"ibm,ddw-applicable"`** property, and implementing the functional extensions specified for the architectural level in Table 128, “DDW Option Extensions,” on page 263. The **`"ibm,ddw-extensions"`** property value is a list of integers the first integer indicates the number of extensions implemented and subsequent integers, one per extension, provide a value associated with that extension. Thus the property value is designed to grow over time in such a way as to enable earlier client programs to ignore later firmware extensions and later client programs to operate on back level firmware. For this level of compatibility to work, the client code needs to ignore extensions beyond what were defined when the client code was written, and be prepared to operate on back level platforms t at do not implement all the extensions that were defined when the client code was written.

Table 128. DDW Option Extensions

DDW Option LoPAPR Level	**`"ibm,ddw-extensions"`** list index	Value Definition
2.7	2	Token of the i*bm,reset-pe-dma-windows* RTAS Call

R1–7.4.10.4–1. For compatibility with changing extensions to the Dynamic DMA Windows (DDW) option: The client program must ignore extensions as represented by **"ibm,ddw-extensions"** value list integers beyond those defined when the client code was written.

R1–7.4.10.4–2. For compatibility with changing extensions to the Dynamic DMA Windows (DDW) option: The client program must be prepared to operate on back level platforms that do not implement all the extensions that were defined when the client code was written, including no extensions at all.

7.4.10.4.1 ibm,reset-pe-dma-windows

The *ibm,reset-dma-windows* call resets the TCE table allocation for the PE to its boot time value as communicated in the **"ibm,dma-window"** OF Device Tree property in the for the PE.

R1–7.4.10.4.1–1. For the Dynamic DMA Windows (DDW) option starting with LoPAPR level 2.7: RTAS must implement the *ibm,reset-dma-windows* call using the argument call buffer defined by Table 129, "ibm,reset-pe-dma-windows Argument Call Buffer," on page 264.

Table 129. *ibm,reset-pe-dma-windows* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for
	Number Inputs	3
	Number Outputs	1
	config_addr	PE configuration address
	PHB_Unit_ID_HI	Represents the most-significant 32-bits of the Unit ID of the PHB that corresponds to the config_addr
	PHB_Unit_ID_Low	Represents the least-significant 32-bits of the Unit ID of the PHB that corresponds to the config_addr
Out	*Status*	0: Success -1: Hardware Error -2: Busy, Try again later -3: Parameter error

R1–7.4.10.4.1–2. For the Dynamic DMA Windows (DDW) option starting with LoPAPR level 2.7: The caller of the *ibm,reset-pe-dma-windows* RTAS call must assure that the TCE table(s) assigned to the PE specified by the *config_addr* field contain no valid mappings at the time of the call (that is, that the window(s) is not being used).

R1–7.4.10.4.1–3. For the Dynamic DMA Windows (DDW) option starting with LoPAPR level 2.7: On a call to *ibm,restore-pe-dma-windows*, the platform must restore the default DMA window per the values provided in the **"ibm,dma-window"** OF Device Tree property in the for the PE (same LIOBN, location, and size).

8 Non-Volatile Memory

This chapter describes the requirements relating to Non-Volatile Memory. Non-Volatile Memory is the repository for system information that must be persistent across reboots and power cycles.

8.1 System Requirements

R1–8.1–1. Platforms must implement at least 8 KB of Non-Volatile Memory. The actual amount is platform dependent and must allow for 4 KB for the OS. Platforms must provide an additional 4 KB for each installed OS beyond the first.

R1–8.1–2. Non-Volatile Memory must maintain its contents in the absence of system power.

R1–8.1–3. Firmware must reinitialize NVRAM to a bootable state if NVRAM data corruption is detected.

R1–8.1–4. OSs must reinitialize their own NVRAM partitions if NVRAM data corruption is detected. OSs may create free space from the first corrupted NVRAM partition header to the end of NVRAM and utilize this area to initialize their NVRAM partitions.

Hardware Implementation Note: The NVRAM terminology used in this chapter goes back to historic implementations that have used battery-powered RAM to implement the non-volatile memory. It should be understood that this is not the only possible implementation. Implementers need to understand that there are no limits on the frequency of writing to the non-volatile memory, so certain technologies may not be applicable. Also, it should be noted that the *nvram-fetch* and *nvram-store* RTAS calls do not allow a "busy" Status return, and this may further limit the implementation choices.

Software Implementation Note: Refer to Section 7.3.1, "NVRAM Access Functions," on page 120 for information on accessing NVRAM.

8.2 Structure

NVRAM is formatted as a set of NVRAM partitions that adhere to the structure in Table 130, "NVRAM Structure," on page 266. NVRAM partitions are prefixed with a *header* containing *signature*, *checksum*, *length*, and *name* fields. The structure of the *data* field is defined by the NVRAM partition creator/owner (designated by *signature* and *name*).

R1–8.2–1. NVRAM partitions must be structured as shown in Table 130, "NVRAM Structure," on page 266.

R1–8.2–2. All NVRAM space must be accounted for by NVRAM partitions.

R1–8.2–3. All IBM-defined NVRAM partitions that are intended to be IBM-unique must have names prefixed with the ASCII representation of the four characters: **`ibm,`**.

Software Implementation Note: Although the data areas of NVRAM partitions are not required to have error checking, it is strongly recommended that the system software implement robust data structures and error checking. Loss of

NVRAM structures due to data corruption can be catastrophic, potentially leading to OS reinstallation and/or complete system initialization.

8.3 Signatures

The *signature* field is used as the first level of NVRAM partition identification. Table 131, “NVRAM Signatures,” on page 267 lists all the currently defined signature types and their ownership classes. The ownership class determines the permission of a particular system software component to create and/or modify NVRAM partitions and/or NVRAM partition contents. All NVRAM partitions may be read by any system software component, but the ownership class has exclusive write permission. Global ownership gives read/write permission to all system software components. These restrictions are made to minimize the possibility of corruption of NVRAM during update activities.

Hardware and Software Implementation Note: It is recommended that NVRAM partitions be ordered on the signature field with the lowest value signature NVRAM partition at the lowest NVRAM address (with the exception of signature = 0x7F, free space). This will minimize the effect of NVRAM data corruption on system operation.

Table 130. NVRAM Structure

Field Name	Size	Description
signature	1 byte	The *signature* field is used to identify the NVRAM partition type and provide some level of checking for overall NVRAM contamination. Signature assignments are given in Table 131, “NVRAM Signatures,” on page 267.

checksum	1 byte	The *checksum* field is included to provide a check on the validity of the header. The checksum covers the *signature*, *length*, and *name* fields and is calculated (on a byte by byte or equivalent basis) by: add, and add 1 back to the sum if a carry resulted as demonstrated with the following program listing.
		
		`unsigned char sumcheck(bp,nbytes)`
		`unsigned char *bp; /* buffer pointer */`
		`unsigned int nbytes; /* number of bytes to sum */`
		`{`
		`unsigned char b_data; /* byte data */`
		`unsigned char i_sum; /* intermediate sum */`
		`unsigned char c_sum; /* current sum */`
		`for (c_sum = 0; nbytes; nbytes--)`
		`{`
		`b_data = *bp++; /* read byte from buffer */`
		`i_sum = c_sum + b_data; /* add to current sum */`
		`if(i_sum < c_sum) /* did a carry out result? */`
		`i_sum += 1; /* if so, add 1 */`
		`c_sum = i_sum; /* copy to current sum */`
		`}`
		`return (c_sum);`
		`}`
		
		This checksum algorithm guarantees 0 to be an impossible calculated value. A valid header cannot have a checksum of zero.

length	2 bytes	The *length* field designates the *total* length of the NVRAM partition, in 16-byte blocks, beginning with the signature and ending with the last byte of the data area. A length of zero is invalid. **Software Implementation Note:** The *length* field must always provide valid offsets to the next header since an invalid length effectively causes the loss of access to every NVRAM partition beyond it.

Table 130. NVRAM Structure ***(Continued)***

Field Name	Size	Description
name	12 bytes	The *name* field is a 12 byte string (or a NULL-terminated string of less than 12 bytes) used to identify a particular NVRAM partition within a signature group. In order to reduce the likelihood of a naming conflict, each platform-specific or OS-specific NVRAM partition name should be prefixed with a company name as specified under the description of the "name" string in the *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], that is, a company name string in one of the three forms described in the reference, followed by a comma (","). If the company name string is null, the name will be interpreted as "other". Before assigning a new name to a NVRAM partition, software should scan the existing NVRAM partitions and ensure that an unwanted name conflict is not created.
data	*length* minus 16 bytes	The structure of the *data* area is controlled by the creator/owner of the NVRAM partition.

Table 131. NVRAM Signatures

Signature (see note)	Signature Type	Ownership Class	# Required	Description
0x70	System	Global	1	For configuration variables.
0x7E	Vendor-defined	Global	0 to n	"name" prefix required.
0x7F	Free Space	Global	0 to n	This signature is used to mark free space in the NVRAM array. The *name* field of all signature 0x7F NVRAM partitions must be set to 0x7...77.
0xA0	OS	Any OS	0 to n	General OS usage.

Note: Any signature not defined above is reserved, and signatures 0x02, 0x50, 0x51, 0x52, 0x71, and 0x72 are reserved for legacy reasons.

8.4 Architected NVRAM Partitions

8.4.1 System (0x70)

System NVRAM partitions are used for storing information (typically, configuration variables) accessible to both OF and the OS. Refer to Appendix B, "LoPAPR Binding," on page 661 for the definition of the contents of the System NVRAM partition named **`common`**.

R1–8.4.1–1. Every system NVRAM must contain a System NVRAM partition with the NVRAM partition name = **`common`**.

R1–8.4.1–2. Data in the **`common`** NVRAM partition must be stored as NULL-terminated strings of the form: <*name*>=<*string*> and the *data* area must be terminated with at least two NULL characters.

R1–8.4.1–3. All *names* used in the **`common`** NVRAM partition must be unique.

R1–8.4.1–4. Device and file specifications used in the **`common`** NVRAM partition must follow IEEE Std 1275 nomenclature conventions.

8.4.1.1 System NVRAM Partition

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

name = value

where *name* follows the rules defined in Section 8.4.1.1.1, “Name,” on page 268 and *value* follows the rules defined in Section 8.4.1.1.2, “Value,” on page 268. The end of the sequence of pairs is denoted by a NULL (0x00) byte.

8.4.1.1.1 Name

Since the data in the System NVRAM partition is an external representation of properties of the **`/option`** node, the name component must follow the rules for *property names* as defined by Section 3.2.2.1.1 Property names of *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2]; i.e., a string of 1-31 printable characters containing no uppercase characters or the characters “/”, “\”, “:”, “[“, “]” or “@”. In addition to these rules, a naming convention is required for OS specific names to avoid name conflicts. Each such name must begin with the OS vendor’s OUI followed by a “,”; e.g., aapl,*xxx* or ibm,*xxx*. This introduces separate name spaces for each vendor in which it manages its own naming conventions.

8.4.1.1.2 Value

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

bnnnnnnn

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

8.4.1.1.3 OF Configuration Variables

OF configuration variables control the operation of OF. In addition to the standard configuration variables defined in *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], other configuration variables are defined by Appendix C, “PA Processor Binding,” on page 753. While such variables are stored in the System NVRAM partition as described above, they have additional rules placed on the format of the value component. Each configuration variable is also represented by a user interface word (of the same name) that returns stack value(s) when that word is evaluated. Each also has a platform defined default value; the absence of a configuration variable in the System NVRAM partition indicates that the value is set to its default value. The format of the external representation of configuration variables, and their stack representation, is defined by Section 7.4.4.1 Configuration Variables of *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2]; the format depends upon the data type of the configuration variable. Whereas the internal storage format is not defined by *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], this architecture specifies them as described below. The names of configuration variables are defined in *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], except as noted otherwise.

8.4.1.1.3.1 Boolean Configuration Variables

The value of a boolean configuration variable is represented in the System NVRAM partition as the string “true” or “false”. The following configuration variables are of type boolean:

auto-boot?

diag-switch?

fcode-debug?

oem-banner?

oem-logo?

use-nvramrc?

little-endian?

real-mode?

menu? (see Section 2.1.3.6, “Boot Process,” on page 44).

8.4.1.1.3.2 Integer Configuration Variables

The value of an integer configuration variable is represented in the System NVRAM partition as a decimal number or a hexadecimal number preceded by “0x”. The following configuration variables are of type integer:

screen-#columns

screen-#rows

security-#badlogins

security-mode

selftest-#megs

real-base

real-size

virt-base

virt-size

load-base

8.4.1.1.3.3 String Configuration Variables

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

boot-command [1]

boot-device [1]

boot-file [1]

diag-device [1]

diag-file [1]

input-device [1]

nvramrc [1]

oem-banner [1]

output-device [1]

security-password [1]

bootinfo-nnnnn [*]

reboot-command [*]

8.4.1.1.3.4 Byte Configuration Variables

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

oem-logo [1]

8.4.1.2 DASD Spin-up Control

In order to reduce the boot time of platforms, a configuration variable is defined to communicate from the platform to the OS to what extent spin-up of hard disk drives can be overlapped. Disk drives generally draw more current as the motors spin up to operating speed, thus the capacity of the power supply limits the ability to spin up drives simultaneously.

The configuration variable **ibm,dasd-spin-interval** indicates the minimum time, in seconds, that must be allowed between initiating the spin-up of hard disk drives on the platform. Presence of this variable potentially allows starting up a drive prior to receiving completion status from a drive previously started. The absence of this variable implies no platform knowledge regarding the capability to overlap and, hence, the OS should wait for the appropriate device status before proceeding to subsequent drives (no overlap).

R1–8.4.1.2–1. If a platform wants to overlap spinning up it's hard disk drives to improve boot performance, it must create the **ibm,dasd-spin-interval** OF configuration variable in the NVRAM signature 0x70 NVRAM partition named **common** and set it equal to an integer that represents the minimum time, in seconds, that must be allowed between initiating the spin-up of drives on the platform.

Firmware Implementation Note: The platform should provide a user-friendly interface to this variable to allow for the possibility of a user installing hard disks that do not conform to the original setting of the variable.

8.4.2 Free Space (0x7F)

R1–8.4.2–1. All unused NVRAM space must be included in a signature = 0x7F Free Space NVRAM partition.

R1–8.4.2–2. All Free Space NVRAM partitions must have the *name* field set to 0x7...77.

8.5 NVRAM Space Management

The only NVRAM partitions whose size an OS can modify are OS and Free Space signature NVRAM partitions. As NVRAM partitions are created and modified by an OS, it is likely that free space will become fragmented; free space consolidation may become necessary.

R1–8.5–1. An OS must not move or delete any NVRAM partition, except OS and Free Space signature NVRAM partitions.

R1–8.5–2. The NVRAM partition header checksum must be calculated as shown in Table 130, “NVRAM Structure,” on page 266.

9 I/O Devices

This chapter describes requirements for IOAs. It adds detail to areas of the PCI architectures (conventional PCI, PCI-X and PCI Express) that are either unaddressed or optional. It also places some requirements on firmware and the OS for IOA support. It provides references to specifications to which IOAs must comply and gives design notes for IOAs that run on LoPAPR systems.

9.1 PCI IOAs

R1–9.1–1. All PCI IOAs must be capable of decoding and generating either a full 32-bit address or a full 64-bit address.

R1–9.1–2. IOAs that implement conventional PCI must be compliant with the most recent version of the *PCI Local Bus Specification* [18] at the time of their design, including any approved Engineering Change Requests (ECRs) against that document.

R1–9.1–3. IOAs that implement PCI-X must be compliant with the most recent version of the *PCI-X Protocol Addendum to the PCI Local Bus Specification* [21] at the time of their design, including any approved Engineering Change Requests (ECRs) against that document.

R1–9.1–4. IOAs that implement PCI Express must be compliant with the most recent version of the *PCI Express Base Specification* [22] at the time of their design, including any approved Engineering Change Requests (ECRs) against that document

Architecture Note: Revision 2.1 and later of the *PCI Local Bus Specification* requires that PCI masters which receive a Retry target termination to unconditionally repeat the same request until it completes. The master may perform other bus transactions, but cannot require those to complete before repeating the original transaction which was previously target terminated with Retry. Revision 2.1 of the specification (page 49) also includes an example which describes how the requirement above applies to a multi-function IOA. See page 48-49 of the 2.1 revision of the *PCI Local Bus Specification* for more detail. Revision 2.0 of the *PCI Local Bus Specification* includes a definition of target termination via Retry, but did not spell out the requirement described above for masters, as does the 2.1 revision of the specification. Masters which are designed based on revision 2.0 of the specification that perform other transactions following target termination with Retry, may cause live-locks and/or deadlocks when installed in a system that utilizes bridges (host bridge or PCI-PCI bridges) that implement Retry, delayed transactions, and/or TCEs, when those masters require following transactions to complete before the original transaction that was terminated with the target Retry. This revision 2.0 to revision 2.1 compatibility problem has been observed on several IOAs that have asked for deviations to Requirement R1–9.1–2. Wording was added to the revision 2.2 of the *PCI Local Bus Specification* which makes a statement similar to this Architecture Note.

9.1.1 Resource Locking

R1–9.1.1–1. PCI IOAs, excepting bridges, must not depend on the PCI LOCK# signal for correct operation nor require any other PCI IOA to assert LOCK# for correct operation.

There are some legacy IOAs on legacy buses which require LOCK#. Additionally, LOCK# is used in some implementations to resolve deadlocks between bridges under a single PHB. These uses of LOCK# are permitted.

9.1.2 PCI Expansion ROMs

R1–9.1.2–1. PCI expansion ROMs must have a ROM image with a code type of 1 for OF as provided in the *PCI Local Bus Specification* [18]. This ROM image must abide by the ROM image format for OF as documented in the *PCI Bus binding to: IEEE Std 1275-1994, Standard for Boot (Initialization, Configuration) Firmware* [6].

LoPAPR systems rely on OF - not BIOS - to boot. This is why strong requirements for OF device support are made.

Vital Product Data (VPD) is an optional feature for PCI adapters and it is strongly recommended that VPD be included in all PCI expansion ROMs. If it is put in the PCI expansion ROM in accordance with the *PCI Local Bus Specification* [18], VPD will be reported in the OF device tree. If the VPD information is formatted as defined in Revision 2.2 with the new capabilities feature, or in any other format, firmware will not read the VPD, and the device driver for the IOA will have to reformat any provided VPD into an OS specified format. It is still required that the keywords and their values must conform to those specified by either PCI 2.1 or PCI 2.2, no matter how they are formatted. Refer to Requirement R1–12.4.2–1.

9.1.3 Assignment of Interrupts to PCI IOAs

R1–9.1.3–1. All PCI IOAs must use the PowerPC interrupt controller, except when made transparent to the OS by the platform through the architected hcall()s.

R1–9.1.3–2. PCI IOAs that do not reside in the Peripheral Memory Space and Peripheral I/O Space of the same PHB must not share the same LSI source.

For further information on the interrupt controller refer to Chapter 6, “Interrupt Controller,” on page 101.

It is strongly advised that system board designers assign one interrupt for each interrupt source. Additionally, multi-function PCI IOAs should have multiple interrupt sources. For restrictions on sharing interrupts with the LPAR option, see Requirement R1–14.4–1. For restrictions on sharing MSIs, see Requirement R1–6.2.3–5 and Requirement R1–6.2.3–6.

9.1.4 PCI-PCI Bridge Devices

R1–9.1.4–1. Firmware must initialize all PCI-to-PCI bridges. See *PCI Bus binding to: IEEE Std 1275-1994, Standard for Boot (Initialization, Configuration) Firmware* [6].

All bridges and switches are required to comply with the bus specification(s) of the buses to which they are attached. See Requirement R1–4.3–1.

9.1.5 Graphics Controller and Monitor Requirements for Clients

The graphics requirements for servers are different from those for portable and personal systems.

R1–9.1.5–1. Plug-in graphics controllers for portable and personal platforms must provide graphics mode sets in the OF PCI expansion ROM image in accordance with the *PCI Bus binding to: IEEE Std 1275-1994, Standard for Boot (Initialization, Configuration) Firmware* [6].

Portable and personal platforms are strongly urged to support some mechanism which allows the platform to electronically sense the display capabilities of monitors.

For graphics controllers that are placed on the system board, the graphics mode sets can be put in system ROM. The mode set software put in the system ROM in this case would be FCode and would be largely or entirely the same as the FCode that would be in the PCI expansion ROM if the same graphics controller was put on a plug-in PCI card.

9.1.6 PCI Plug-in Graphic Cards

R1–9.1.6–1. *(Requirement Number Reserved For Compatibility)*

R1–9.1.6–2. PCI plug-in graphics cards which are going to be the primary display IOA during the time prior to the OS device driver being loaded must contain an OF display driver on the IOA.

9.1.7 PCI Cache Support Protocol

The PCI architecture allows for the optional implementation of caching of data. This architecture basically assumes that the data in I/O memory is non-coherent. As such, platforms are not required to implement the optional PCI Cache Support protocol using the SBO# and SDONE signals. Therefore, IOAs used in LoPAPR platforms should not count on those signals for proper operations.

R1–9.1.7–1. IOAs used in LoPAPR platforms and their device drivers must not require the use of the PCI signals SBO# and SDONE for proper operations.

9.1.8 PCI Configuration Space for IOAs

There are several writable fields in the PCI Configuration Header. Some of these are written by the firmware and should never be changed by the device driver.

R1–9.1.8–1. All registers and bits in the PCI Configuration Header must be set to a platform specific value by firmware and preserved by software, except that software is responsible for setting the configuration space as indicated in Table 132, “Software Programming of PCI Configuration Header Registers,” on page 275.

Table 132. Software Programming of PCI Configuration Header Registers

Register Name	Bit Name	Software Action
Command	Bus Master	Must write to a 1 before the first DMA operation after a reset. Must write to a 0 before unconfiguring device driver.
	Memory Space	Must write to a 1 before the first MMIO operation to IOA’s memory space (if any) after a reset. Must write to a 0 before unconfiguring device driver.
	IO Space	Must write to a 1 before the first MMIO operation to IOA’s I/O space (if any) after a reset. Must write to a 0 before unconfiguring device driver.
	all other bits	Must restore to previous value after any reset operation (for example, via *ibm,set-slot-reset Function* 1 or 3). The *ibm,configure-bridge* RTAS call is available to assist in restoring values, where appropriate.
Built in Self Test (BIST)	all	If implemented, software may use if desired.
all other PCI header registers that may be modified by firmware after initial reset or by *ibm,configure-connector* for DR operations	all	Must restore to previous value after any reset operation (for example, via *ibm,set-slot-reset-state Function* 1). The *ibm,configure-bridge* RTAS call is available to assist in configuring PCI bridges and switches, where appropriate.

R1–9.1.8–2. All IOAs that implement PCI-X Mode 2 or PCI Express must supply the **`“ibm,pci-config-space-type”`** property (see Section B.6.5.1.1.1, “Properties for Children of PCI Host Bridges,” on page 703).

Implementation Note: The **"ibm,pci-config-space-type"** property in Requirement R1–9.1.8–2 is added for platforms that support I/O fabric and IOAs that implement PCI-X Mode 2, and PCI Express. To access the extended configuration space provided by PCI-X Mode 2 and PCI Express, all I/O fabric leading up to an IOA must support a 12-bit register number. In other words, if a platform implementation has a conventional PCI bridge leading up to an IOA that implements PCI-X Mode 2, the platform will not be able to provide access to the extended configuration space of that IOA. The **"ibm,config-space-type"** property in the IOA's OF node is used by device drivers to determine if an IOA's extended configuration space can be accessed.

9.1.9 PCI IOA Use of PCI Bus Memory Space Address 0

Some PCI IOAs will fail when given a bus address of 0. In the PC world, address 0 would not be a good address, so some PCI IOA designs which were designed for the PC arena will check for an address of 0, and fail the operation if it is 0.

R1–9.1.9–1. For systems that use PCI IOAs which will fail when given a bus address of 0 for DMA operations, and when the operations for which those IOAs are used are other than system memory dump operations, then the OS must prevent the mapping of PCI bus address 0 for PCI DMA operation for such IOAs.

R1–9.1.9–2. PCI IOAs used for dumping contents of system memory must operate properly with a PCI bus address of 0 for PCI DMA operations.

R1–9.1.9–3. The firmware must not map an IOA used for loading a boot image to an address of 0, when loading a boot image, if that IOA cannot accept an address of 0.

Implementation Note: A reasonable implementation of Requirement R1–9.1.9–1 would be to have an interface between the device driver and the kernel to allow the device driver to indicate to the kernel that the restriction is required for that IOA, so that all IOAs for that kernel image are not affected.

9.1.10 PCI Express Completion Timeout

Prior to the implementation of the PCI Express additional capability to set the Completion Timeout Value and Completion Timeout Disable in the PCI Express Device Control 2 register of an IOA, the IOAs need device-specific way to provide the disable capability. In addition, the platforms need to provide a way for the OSs and device drivers to know when to disable the completion timeout of these devices that only provide a device-specific way of doing so.

R1–9.1.10–1. PCI Express IOAs must either provide a device-specific way to disable their DMA Completion Timeout timer or must provide the Completion Timeout Disable or Completion Timeout Value capability in the PCI Express Device Control 2 register, and device drivers for IOAs that provide a device-specific way must disable their DMA Completion Timeout timer if it is either unknown whether the IOA provides a sufficiently long timer value for the platform, or if it is known that they do not provide a sufficient timeout value (for example, if the **"ibm,max-completion-latency"** property is not provided).

R1–9.1.10–2. Platforms must provide the **"ibm,max-completion-latency"** property in each PCI Express PHB node of the OF Device Tree.

9.1.11 PCI Express I/O Virtualized (IOV) Adapters

PCI Express defines I/O Virtualized (IOV) adapters, where such an adapter has separate resources for each virtual instance, called a Virtual Function (VF). There are two PCI specifications that exist to define such adapters:

- *Single Root I/O Virtualization and Sharing Specification* [29] defines the requirements for SR-IOV adapters.

- *Multi-Root I/O Virtualization and Sharing Specification* [30] defines the requirements for MR-IOV adapters.

The interface presented to an OS from an MR-IOV adapter will look the same as an SR-IOV adapters, and therefore will not be described separately here.

IOV adapters and/or the VFs of an IOV adapter that has IOV enabled, are assigned to OSs as follows (see also Table 133, “IOV Environment Characteristics,” on page 277 for a full set of characteristics of these environments):

- For the *Legacy Dedicated* environment, the entire adapter is assigned to one LPAR, with the IOV functionality not enabled. In this mode, the OS provides device driver(s) for the adapter Function(s). VFs do not exist, because IOV is not enabled. The OS is given the capability to do Hot Plug add, remove, and replace in a non-managed environment (without an HMC), and may be given that capability in a managed environment.
- For the *SR-IOV Non-shared* environment, the entire adapter is assigned to one LPAR, with IOV functionality enabled, but with the Physical Function(s) (PFs) of the adapter hosted by the platform. Only VFs are presented to the OS. The OS is given the capability to do Hot Plug add, remove, and replace in a non-managed environment (without an HMC), and may be given that capability in a managed environment.
- For the *SR-IOV Shared* environment, the adapter is assigned to the platform, with IOV functionality enabled. The platform then assigns VF(s) to OS(s). Only the managed environment applies, and add/remove/replace operations are controlled by DLPAR operations to the OS(s) from the management console.

For all environments except the SR-IOV Shared, multiple functions will appear as a multi-function IOA with possible sharing of a single PE. For example, the multi-function adapters may have a shared EEH domain and shared DMA window.

Determination of which of the above environments is supported for a given platform and partition or OS type is beyond the scope of this architecture.

Table 133, “IOV Environment Characteristics,” on page 277 defines the characteristics of these environments.

Table 133. IOV Environment Characteristics

	Legacy Dedicated	SR-IOV Non-shared	SR-IOV Shared
Entire adapter assigned to OS, IOV not enabled	yes	n/a	n/a
Entire adapter assigned to OS, IOV enabled	n/a	yes	n/a
Adapter can be shared across multiple OSs, IOV enabled	n/a	n/a	yes
Function DD support	Plain Function only (not VF or PF)	VF only	VF only
PFs managed by platform?	n/a	yes	yes
Managed environment support?	yes	yes	yes
Non-managed environment support?	yes	yes	no
OS controlled Hot Plug capable?	yes	yes	no
DLPAR capable?	yes	yes	yes
All functions under one PHB in the OF Device Tree for the adapter?	yes	yes	no
All functions under separate PHBs in the OF Device Tree for the same adapter[a]?	no	no	yes
config_addr translation (virtualization) by the platform (that is, the bus/device/function of the *config_addr* does not necessarily correspond to what the device has programmed)	no	yes	yes

Table 133. IOV Environment Characteristics ***(Continued)***

	Legacy Dedicated	SR-IOV Non-shared	SR-IOV Shared
Shared PE domain (for example, shared EEH domain, shared DMA window)	yes	yes	no

a. The adapter is physically under one PHB, but the platform creates separate "virtual" PHBs in the OF Device Tree and virtualizes the PCI Express configuration space for the various functions.

R1–9.1.11–1. PCI Express Single Root IOV (SR-IOV) adapters must comply to the *Single Root I/O Virtualization and Sharing Specification* [29].

R1–9.1.11–2. PCI Express Multi-Root IOV (MR-IOV) adapters must comply to the *Multi-Root I/O Virtualization and Sharing Specification* [30].

R1–9.1.11–3. The platform must present within the device tree nodes for all PCI Express adapters configured to operate in IOV mode the **`"ibm,is-vf"`** property as defined in section B.6.5.1.1.2, "LPAR Option Properties," on page 706.

9.2 Multi-Initiator SCSI Support

Multi-initiator SCSI support is identified in the OF device tree.

R1–9.2–1. Platform Implementation: Platforms must support the **`"scsi-initiator-id"`** property as described in Appendix B, "LoPAPR Binding," on page 661 and *Open Firmware Recommended Practice: Device Support Extensions* [5].

9.3 Contiguous Memory

I/O devices that require contiguous memory pages (either real or via contiguous TCEs) cannot reasonably be accommodated in LoPAPR platforms. When TCEs are turned off, that would require that real physical memory addresses be allocated. When TCEs are on, that would require contiguous TCEs be assigned, and although that is the first attempt by the OS's TCE assignment algorithm, the algorithm will assign non-contiguous ones if contiguous ones cannot be assigned. Dynamic Reconfiguration complicates the contiguous problem even further.

R1–9.3–1. I/O devices and/or their device drivers used in LoPAPR platforms must implement scatter/gather capability for DMA operations such that they do not require contiguous memory pages to be allocated for proper operation.

9.4 Re-directed Serial Ports

The **`"ibm,vty-wrap-capable"`** OF device tree property will be present in an OF device tree of a serial port node when the OS data communication with that serial port controller can be redirected, or wrapped, away from the physical serial port connector to an `ibm,vty` device, which is often a virtual terminal session of the Hardware Management Console (HMC). This property indicates to serial port diagnostic programs that additional end user information should be displayed during the serial port diagnostic test indicating that it is possible that serial port data could be redirected away from the physical serial port preventing the execution of wrap tests with physical wrap plugs. The end user information should describe that initiating a virtual terminal session causes the serial port controller's data to be wrapped away from the physical serial port connection and that terminating a virtual terminal session causes the serial port controller's data to be returned to the physical serial port connection. The **`"ibm,vty-wrap-capable"`** property is present with a value of null when this re-direction capability exists and is absent when this capability does not exist.

R1–9.4–1. The **"ibm,vty-wrap-capable"** OF device tree property must be present in an OF device tree of a serial port node when the OS data communication with that serial port controller can be redirected, or wrapped, away from the physical serial port connector to an ibm,vty device, and must not be present if this capability does not exist.

9.5 System Bus IOAs

This section lists the requirements for the systems to support IOAs connected to the system bus or main I/O expansion bus.

R1–9.5–1. Each system bus IOA must be a bus master.

R1–9.5–2. Firmware must assign unique addresses to all system bus IOA facilities.

R1–9.5–3. Addresses assigned to system bus IOA facilities must not conflict with the addresses mapped by any host bridge on the system bus.

R1–9.5–4. System bus IOAs must be assigned interrupt sources for their interrupt requirements by firmware.

R1–9.5–5. A system bus IOA's OF **"interrupts"** property must reflect the interrupt source and type allocation for the device.

R1–9.5–6. All system bus IOA interrupts must be low true level sensitive (referred to as level sensitive).

R1–9.5–7. Interrupts assigned to system bus IOAs must not be shared with other IOAs.

R1–9.5–8. The OF unit address (first entry of the **"reg"** property) of a system bus IOA must stay the same from boot to boot.

R1–9.5–9. Each system bus IOA must have documentation for programming the IOA and an OF binding which describes at least the **"name"**, **"reg"**, **"interrupts"**, and **"interrupt-parent"** properties for the device.

10 Error and Event Notification

10.1 Introduction

RTAS provides a mechanism which helps OSs avoid the need for platform-dependent code that checks for, or recovers from, errors or exceptional conditions. The mechanism is used to return information about hardware errors which have occurred as well as information about non-error events, such as environmental conditions (for example, temperature or voltage out-of-bounds) which may need OS attention. This permits RTAS to pass hardware event information to the OS in a way which is abstracted from the platform hardware. This mechanism primarily presents itself to the OS via two RTAS functions, *event-scan* and *check-exception*, which are described further in Section 7.3.3, "Error and Event Reporting," on page 125. A further RTAS function, *rtas-last-error*, is also provided to return information about hardware failures detected specifically within an RTAS call.

The *event-scan* function is called periodically to check for the presence or past occurrence of a hardware event, such as a soft failure or voltage condition, which did not cause a program exception or interrupt (for example, an ECC error detected and corrected by background scrubbing activity). The *check-exception* function is called to provide further detail on what platform event has occurred when certain exceptions or interrupts are signaled. The events reported by these two functions are mutually exclusive on any given platform; that is, a platform may choose to notify the OS of a particular event type either through *event-scan* or through an interrupt and *check-exception*, but not both.

Since firmware is platform-specific, it can examine hardware registers, can often diagnose many kinds of hardware errors down to a root cause, and may even perform some very limited kinds of error recovery on behalf of the OS. The reporting format, described in this chapter, permits firmware to report the type of error which has occurred, what entities in the platform were involved in the error, and whether firmware has successfully recovered from the error without the need for further OS involvement. Firmware may not, in many cases, be able to determine all the details of an error, so there are also returned values which indicate this fact. Firmware may optionally provide extended error diagnostic information, as described in Section 10.3.2.2, "Version 6 Extensions of Event Log Format," on page 294.

The abstractions provided by this architecture enable the handling of most platform errors and events without integrating platform-specific code into each supported OS.

Architecture Note: It is not a goal of the firmware to diagnose all hardware failures. Most I/O device failures, for example, will be detected and recovered by an associated device driver. Firmware attempts to determine the cause of a problem and report what it finds, to aid the end user (by providing meaningful diagnostic data for messages) and to prevent the loss of error syndrome information. Firmware is never required to correct any problem, but in some cases may attempt to do so. System vendors who want more extensive error diagnosis may create OS error handlers which contain specific hardware knowledge, or could use firmware to collect a minimum set of error information which could then be used by diagnostics to further analyze the cause of the error.

10.2 RTAS Error and Event Classes

Table 134, "Error and Event Classes with RTAS Function Call Mask," on page 282 describes the predefined classes of error and event notifications that can be presented through the *check-exception* and *event-scan* RTAS functions. More detailed descriptions of these classes are given later in this chapter. Table 134, "Error and Event Classes with RTAS Function Call Mask," on page 282 defines nodes in the OF device tree which, through an **`"interrupts"`** property, may list the platform-dependent interrupts related to each class. From this information, OSs know which interrupts may be handled by calling *check-exception*. The OF structure for describing these interrupts is defined in Appendix B,

"LoPAPR Binding," on page 661. Table 134, "Error and Event Classes with RTAS Function Call Mask," on page 282 also defines the mask parameter for the *check-exception* and *event-scan* RTAS functions which limits the search for errors and events to the classes specified.

Table 134. Error and Event Classes with RTAS Function Call Mask

Class Type	OF Node Name (where the **"interrupts"** property lists the interrupts)	RTAS Function Call Mask (value = 1 enables class)
Internal Errors	internal-errors	bit 0
Environmental and Power Warnings	epow-events	bit 1
Reserved		bit 2
Hot Plug Events	hot-plug-events	bit 3
I/O Events and Errors	ibm,io-events	bit 4

R1–10.2–1. For the Platform Interrupt Event option: The platform must implement the I/O Events and Errors class type along with the appropriate **ibm,io-events** node property to specify the interrupts.

R1–10.2–2. Platform-specific error and event interrupts that a platform provider wants the OS to enable must be listed in the **"interrupts"** property of the appropriate OF event class node, as described in Table 134, "Error and Event Classes with RTAS Function Call Mask," on page 282.

R1–10.2–3. To enable platform-specific error and event interrupt notification, OSs must find the list of interrupts (described in Table 134, "Error and Event Classes with RTAS Function Call Mask," on page 282) for each error and event class in the OF device tree, and enable them.

R1–10.2–4. OSs must have interrupt handlers for the enabled interrupts described in Requirement R1–10.2–3, which call the RTAS *check-exception* function to determine the cause of the interrupt.

R1–10.2–5. Platforms which support error and event reporting must provide information to the OS via the RTAS *event-scan* and *check-exception* functions, using the reporting format described in Table 137, "RTAS Event Return Format (Fixed Part)," on page 292.

R1–10.2–6. Optional Extended Error Log information, if returned by the *event-scan* or *check-exception* functions, must be in the reporting format described in Table 138, "RTAS General Extended Event Log Format, Version 6," on page 296.

R1–10.2–7. To provide control over performance, the RTAS event reporting functions must not perform any event data gathering for classes not selected in the event class mask parameter, nor any extended data gathering if the time critical parameter is non-zero or the log buffer length parameter does not allow for an extended error log.

R1–10.2–8. To prevent the loss of any event notifications, the RTAS event reporting functions must be written to gather and process error and event data without destroying the state information of events other than the one being processed.

R1–10.2–9. Any interrupts or interrupt controls used for error and event notification must not be shared between error and event classes, or with any other types of interrupt mechanisms. This allows the OS to partition its interrupt handling and prevents blocking of one class of interrupt by the processing of another.

R1–10.2–10. If a platform chooses to report multiple event or error sources through a single interrupt, it must ensure that the interrupt remains asserted or is re-asserted until *check-exception* has been used to process all outstanding errors or events for that interrupt.

Platform Implementation Note: In Requirement R1–10.2–5, although the fixed-part return format for *check-exception* and *event-scan* is the same, there are some expectations about what types of error response may be returned from these functions, as follows:

- The *event-scan* function is mainly intended to report only errors that have been recovered or are non-critical to the OS, since it is only called on a periodic basis. As such, it should never be used to report a Severity greater than "WARNING". More critical errors should be signaled by an interrupt. Typically, the expected response of an OS to an *event-scan* error report is simply to log the error. The *check-exception* function may report error information of any severity.
- If *event-scan* is reporting a critical error (for example, a checkstop) that occurred before the current boot session, it should not report it with a "FATAL" Severity, even though the condition was fatal at the time the failure occurred. The Severity field informs the OS of the severity of the event at the time of reporting. Errors which occurred before a successful reboot are no longer critical. Likewise, the RTAS Disposition field for such an error should be "FULLY_RECOVERED". There is a bit in the extended error log to indicate these "residual" errors.
- Although *check-exception* can potentially clean up an error and return a "FULLY_RECOVERED" disposition, recovery still may not occur if the MSR_{RI} bit is not set to 1. It is up to the OS to examine the RI bit, to determine whether processor state is preserved so that a return from the machine check interrupt handler can be safely attempted.

10.2.1 Internal Error Indications

Hardware may detect a variety of problems during operation, ranging from soft errors which have already been corrected by the time they are reported, to hard errors of such severity that the OS (and perhaps the hardware) cannot meaningfully continue operation. The mechanisms described in Section 10.1, "Introduction," on page 281 are used to report such errors to the OS. This section describes the architectural sources of errors, and describes a method that platforms can use to report the error. All OSs need to be prepared to encounter the errors reported as they are described here. However, in some platforms more sophisticated handling may be introduced via RTAS, and the OS may not have to handle the error directly. More robust error detection, reporting, and correcting are at the option of the hardware vendor.

The primary architectural mechanism for indicating hardware errors to an OS is the machine check interrupt. If an error condition is surfaced by placing the system in checkstop, it precludes any immediate participation by the OS in handling the error (that is, no error capture, logging, recovery, analysis, or notification by the OS). For this reason, the machine check interrupt is preferred over going to the checkstop state. However, checkstop may be necessary in certain situations where further processing represents an exposure to loss of data integrity. To better handle such cases, a special hardware mechanism may be provided to gather and store residual error data, to be analyzed when the system goes through a subsequent successful reboot.

Less critical internal errors may also be signaled to the OS through a platform-specific interrupt in the "Internal Errors" class, or by periodic polling with the *event-scan* RTAS function.

Architecture Note: The machine check interrupt will not be listed in the OF node for the "Internal Errors" class, since it is a standard architectural mechanism. The machine check interrupt mechanism is enabled from software or firmware by setting the MSR_{ME} bit =1. Upon the occurrence of a machine check interrupt, bits in SRR1 will indicate the source of the interrupt and SRR0 will contain the address of the next instruction that would have been executed if the interrupt had not occurred. Depending on where the error is detected, the machine check interrupt

may be surfaced from within the processor, via logical connection to the processor machine check interrupt pin, or via a system bus error indicator (for example, Transfer Error Acknowledge - TEA).

R1–10.2.1–1. OSs must set MSR_{ME}=1 prior to the occurrence of a machine check interrupt in order to enable machine check processing via the *check-exception* RTAS function.

Architecture Note: Requirement R1–10.2.1–1 is not applicable when the FWNMI option is used.

R1–10.2.1–2. For hardware-detected errors, platforms must generate error indications as described in Table 135, "Error Indications for System Operations," on page 285, unless the error can be handled through a less severe platform-specific interrupt, or the nature of the error forces a checkstop condition.

R1–10.2.1–3. Platforms which detect and report the errors described in Table 135, "Error Indications for System Operations," on page 285 must provide information to the OS via the RTAS *check-exception* function, using the reporting format described in Table 137, "RTAS Event Return Format (Fixed Part)," on page 292.

R1–10.2.1–4. To prevent error propagation and allow for synchronization of error handling, all processors in a multi-processor system must receive any machine check interrupt signaled via the external machine check interrupt pin.

Platform Implementation Notes:

1. The intent of Requirement R1–10.2.1–2 is to define standard error notification mechanisms for different hardware error types. For most hardware errors, the signaling mechanism is the machine check interrupt, although this requirement hints at the use of a less severe platform-specific interrupt for some errors. The important point here is actually whether the error can be handled through that interrupt. Simply using an external interrupt to signal the error is not sufficient. The hardware and RTAS also need to limit the propagation of corrupted data, prevent loss of error state data, and support the cleanup and recovery of such an error. Since the response to an external interrupt may be significantly slower than a machine check, and in fact may be masked, the error should not require immediate action on the part of the OS and/or RTAS. In addition, external interrupts (except external machine check interrupts) are reported to only one processor, so operations by the other processors in an MP system should not be impacted by this error unless they specifically try to access the failing hardware element. To summarize, platforms should not use platform-specific interrupts to signal hardware errors unless there is a complete hardware/RTAS platform solution for handling such errors.

2. The intent of Requirement R1–10.2.1–4 is that most hardware errors would be signaled simultaneously to all processors, so that processors could synchronize the error handling process; that is, one processor would be chosen to do the call to *check-exception*, while the other processors remained idle so that they would not interfere with RTAS while it gathered machine check error data. While this is a straightforward wiring solution for errors signaled via the external machine check interrupt pin, that is not the case for internal processor errors or processor bus errors. Typically, only one processor will see such errors. An internal processor error can be identified with just the contents of SRR1, and so can be handled without synchronization with other processors. Processor bus errors, however, can be more difficult, especially if the error is propagated up to the processor bus from a lower-level bus. In general, such propagation should be avoided, and such errors should be signaled through the machine check interrupt pin to ensure proper error handling.

10.2.1.1 Error Indication Mechanisms

Table 135, "Error Indications for System Operations," on page 285 describes the mechanisms by which software will be notified of the occurrence of operational failures during the types of data transfer operations listed below. The assumption here is that the error notification can occur only if a hardware mechanism for error detection (for example, a parity checker) is present. In cases where there is no specific error detection mechanism, the resulting condition, and whether the software will eventually recognize that condition as a failure, is undefined.

Table 135. Error Indications for System Operations

Initiator	Target	Operation	Error Type (if detected)	Indication to Software	Comments
Processor	N/A	Internal	Various	Machine check	Some may cause checkstop
Processor	Memory	Load	Invalid address	Machine check	
			System bus time-out	Machine check	
			Address parity on system bus	Machine check	
			Data parity on system bus	Machine check	
			Memory parity or uncorrectable ECC	Machine check	
		Store	Invalid address	Machine check	
			System bus time-out	Machine check	
			Address parity on system bus	Machine check	
			Data parity on system bus	Machine check	
		External cache load	Memory parity or uncorrectable ECC	Machine check	Associated with Instruction Fetch or Data Load
		External cache flush	Cache parity or uncorrectable ECC	Machine check	
		External cache access	Cache parity or uncorrectable ECC	Machine check	Associated with Instruction Fetch or Data Transfer
Processor	I/O	Load or Store	Various errors between the processor and the I/O fabric	Machine check	I/O fabrics include hubs and bridges and interconnecting buses or links.
Processor	I/O bus configuration space	Read	Various, except no response from IOA	Firmware receives a machine check, OS receives all=1's data along with a *Status* of -1 from the RTAS call	If EEH is implemented and enabled, firmware does not get a machine check and the PE is in the EEH Stopped State on return from the RTAS call
			No response from an IOA	All-1's data returned, along with a "Success" *Status* from the RTAS call	If EEH is implemented and enabled, the PE is not in the EEH Stopped State on return from the RTAS call
		Write	Various, except no response from IOA	Firmware receives a machine check, OS receives a *Status* of -1 from the RTAS call	If EEH is implemented and enabled, firmware does not get a machine check and the PE is in the EEH Stopped State on return from the RTAS call
			No response from an IOA	Operation ignored, OS receives a "Success" *Status* from the RTAS call	If EEH is implemented and enabled, the PE is in the EEH Stopped State on return from the RTAS call

Table 135. Error Indications for System Operations ***(Continued)***

Initiator	Target	Operation	Error Type (if detected)	Indication to Software	Comments
Processor	I/O bus; I/O Space or Memory Space	Load	Various, except no response from IOA	Machine check	If EEH is implemented and enabled, no machine check is received, all-1's data is returned, and the PE enters the EEH Stopped State
			No response from an IOA	All-1's data returned	Invalid address, broken IOA, or configuration cycle to non-existent IOA; if EEH is implemented and enabled, the PE enters the EEH Stopped State
		Store	Various, except no response from IOA	Machine check	If EEH is implemented and enabled, no machine check is received and the PE enters the EEH Stopped State
			No response from IOA	Ignore Store	Invalid address, broken IOA, or configuration cycle to non-existent IOA; if EEH is implemented and enabled, the PE enters the EEH Stopped State
Processor	Invalid address (addressing an "undefined" address area)	Load or Store	No response from system	Machine check	
I/O	Memory	DMA - either direction	Various, including but not limited to: ◆ Invalid addr accepted by a bridge ◆ TCE extent ◆ TCE Page Mapping and Control mis-match or invalid TCE	Machine check unless reportable directly to the IOA in a way that allows the IOA to signal the error to its device driver	If EEH is implemented and enabled, no machine check is received and the PE enters the EEH Stopped State
I/O	I/O	DMA - either direction	Various	Machine check unless reportable to master of the transfer in a way that allows master to recover	
I/O	Invalid address	DMA - either direction	No response from any IOA	PCI IOA master-aborts	Signal device driver using an external interrupt
PCI IOA	-	Any	Internal, indicated by SERR# or ERR_FATAL	SERR# or ERR_FATAL, causing machine check	If EEH is implemented and enabled, no machine check is received and the PE enters the EEH Stopped State

Implementation Note: IOAs should, whenever possible, detect the occurrence of PCI errors on DMA and report them via an external interrupt (for possible device driver recovery) or retry the operation. Since system state has not been lost, reporting these errors via a machine check to the CPUs is inappropriate. Some devices or device drivers

may cause a catastrophic error. Systems which wish to recover from these types of errors should choose devices and device drivers which are designed to handle them correctly.

10.2.2 Environmental and Power Warnings

Environmental and Power Warnings (EPOW) is an option that provides a means for the platform to inform the OS of these types of events. The intent is to enable the OS to provide basic information to the user about environmental and power problems and to minimize the logical damage done by these problems. For example, an OS might want to abort all disk I/O operations in progress to ensure that disk sectors are not corrupted by the loss of power. Even on platforms that provide hardware protection of data during environmental events, EPOW notification allows discrimination between I/O errors caused by hardware failures versus EPOW events.

These warnings include action codes that the platform can use to influence the OS behavior when various hardware components fail. For example, a fan failure where the system can continue to operate in the safe cooling range may just generate an action code of WARN_COOLING, but a fan failure where the system cannot operate in the safe cooling range may generate an action code of SYSTEM_HALT.

Implementation Note: Hardware cannot assume that the OS will process or take action on these warnings. These warnings are only provided to the OS in order to allow the OS a chance to cleanly abort operations in progress at the time of the warning. Hardware still assumes responsibility for preventing hardware damage due to environmental or power problems.

An OS that wants to be EPOW-aware will look for the **`epow-events`** node in the OF device tree, enable the interrupts listed in its **`"interrupts"`** property, and provide an interrupt handler to call *check-exception* when one of those interrupts are received.

When an EPOW event occurs, whether reported by *check-exception* or *event-scan*, RTAS will directly pass back the EPOW sensor value as part of the Extended Error Log format as described in Table 146, “Platform Event Log Format, Version 6, EPOW Section,” on page 308, assuming the extended log is requested. Doing so avoids the need for the OS to make an extra RTAS call to obtain the sensor value. For critical power problems, the *check-exception* function is used to immediately report changes of state to the OS, while the *get-sensor-state* function allows the OS to monitor the condition (for example, loss of AC power) to see if the problem corrects itself.

R1–10.2.2–1. If the platform supports Environmental and Power Warnings by including a EPOW device tree entry, then the platform must support the EPOW sensor for the *get-sensor-state* RTAS function.

R1–10.2.2–2. The EPOW sensor, if provided, must contain the EPOW action code (defined in Table 136, “EPOW Action Codes,” on page 288) in the least significant 4 bits. In cases where multiple EPOW actions are required, the action code with the highest numerical value (where 0 is lowest and 7 is highest) must be presented to the OS. The platform may implement any subset of these action codes, but must operate as described in Table 136, “EPOW Action Codes,” on page 288 for those it does implement.

R1–10.2.2–3. To ensure adequate response time, platforms which implement the EPOW_MAIN_ENCLOSURE or EPOW_POWER_OFF action codes must do so via interrupt and *check-exception* notification, rather than by *event-scan* notification. ***(Except as modified by Requirement R1–10.2.2–4)***

R1–10.2.2–4. If the platform does not notify EPOW_MAIN_ENCLOSURE or EPOW_POWER_OFF via interrupt, then the platform must protect data on I/O storage devices from corruption due to the EPOW event.

R1–10.2.2–5. For interrupt-driven EPOW events, the platform must ensure that an EPOW interrupt is not lost in the case where a numerically higher-priority EPOW event occurs between the time when *check-exception* gathers the sensor value and when it resets the interrupt.

R1–10.2.2–6. For SYSTEM_SHUTDOWN EPOW class 3, after a SYSTEM_SHUTDOWN EPOW commences and when the delay interval timer expires, if an **`"ibm,recoverable-epow3"`** encode-null property in the **`/rtas`** node is present, then the OS code that manages preserving storage must check the EPOW sensor

state and the **"ibm,request-partition-shutdown"** property if present. A normal boot must only occur when the EPOW sensor state indicates that the EPOW condition requiring a shutdown no longer exists (EPOW 0) and the **"ibm,request-partition-shutdown"** is not present. Otherwise, the code that manages preserving storage must take the action as identified by the property.

Implementation Note: One way for hardware to prevent the loss of an EPOW interrupt is by deferring the generation of a new EPOW interrupt until the existing EPOW interrupt is reset by a call to the RTAS *check-exception* function. Another way is to ignore resets to the interrupt until all EPOW events have been reported.

Table 136. EPOW Action Codes

Action Code	Value	Description
EPOW_RESET/MESSAGE	0	No EPOW event is pending. This action code is the lowest priority.
WARN_COOLING	1	A non-critical cooling problem exists. An EPOW-aware OS logs the EPOW information.
WARN_POWER	2	A non-critical power problem exists. An EPOW-aware OS logs the EPOW information.
SYSTEM_SHUTDOWN	3	The system must be shut down. An EPOW-aware OS logs the EPOW error log information, then schedules the system to be shut down to begin after an OS defined delay internal (default is 10 minutes.)
SYSTEM_HALT	4	The system must be shut down quickly. An EPOW-aware OS logs the EPOW error log information, then schedules the system to be shut down in 20 seconds.
EPOW_MAIN_ENCLOSURE	5	The system may lose power. The hardware ensures that at least 4 milliseconds of power within operational thresholds is available after signalling an interrupt. An EPOW-aware OS performs any desired functions, masks the EPOW interrupt, and monitors the sensor to see if the condition changes. Hardware does not clear this action code until the system resumes operation within safe power levels.
EPOW_POWER_OFF	7	The system will lose power. The hardware ensures that at least 4 milliseconds of power within operational thresholds is available after signalling an interrupt. An EPOW-aware OS performs any desired operations, then attempts to turn system power off. An EPOW-aware OS does not clear the EPOW interrupt for this action code. This action code is the highest priority.

Software Implementation Note: A recommended OS processing method for an EPOW_MAIN_ENCLOSURE event is as follows:

- Prepare for shutdown, mask the EPOW interrupt, and wait for 50 milliseconds. Then call *get-sensor-state* to read the EPOW sensor.
- If the EPOW action code is unchanged, wait an additional 50 milliseconds.
- If the action code is EPOW_POWER_OFF, attempt to power off. Otherwise, the power condition may have stabilized, so interrupts may be enabled and normal operation resumed.

Implementation Note: EPOW_RESET (EPOW action code 0) may be used to indicate that a previously reported EPOW condition is no longer present. For instance, a system might see a WARN_POWER action code for a loss of a redundant line input power. EPOW_RESET may subsequently be issued if the line power were restored. The same bits in the EPOW error log that specified the type of WARN_POWER EPOW generated would be set in the EPOW_RESET error log to indicate the specific EPOW event that was reset.

Systems that do not support an EPOW interrupt would generally be unable to support the EPOW action codes 5 and 7. In those cases, there could not be an EPOW event to indicate a loss of power. However, after power were restored, generating the EPOW_RESET EPOW would indicate that the system had lost power previously and the

power had been restored. The EPOW_RESET should only be used in this way if the system is unable to generate an EPOW class 5 or class 7.

10.2.3 Hot Plug Events

Hot Plug Events, when implemented, are reported through the *event-scan* RTAS call. These events are surfaced through the fixed portions of the RTAS return value. (see Table 137, “RTAS Event Return Format (Fixed Part),” on page 292) Some parts of the system may be modified without direct support from the OS.

R1–10.2.3–1. If FRUs can be hot plugged in the system without OS support, the Hot Plug Event mechanism must be provided for signaling the OS about the event.

10.3 RTAS Error and Event Information Reporting

Architecture Note: All data formats listed in this section are either referenced as byte fields (and therefore are independent of Endian orientation), or an indicator in the data structure describes their Endian orientation. Bits are numbered from left (high-order:0) to right (low-order:7).

10.3.1 Introduction

This section describes the data formats used to report events and errors from RTAS to the OS. A common format is used for errors and events to simplify software both in RTAS and in the OS. Both errors and events may have been analyzed to some degree by RTAS, and value judgments may have been applied to decide how serious an error is, or even how to describe it to the OS. These judgments are made by platform providers, since only they know enough about the hardware to decide how serious a problem it is, whether and how to recover from it, and how to map it onto the abstracted set of events and errors that a system is required to know about. There will be cases with some platforms where no reasonable mapping exists, and platform features may not be fully supported by the OS. In such cases, error reports may also be non-specific, leaving platform-specific details to platform-aware software.

10.3.2 RTAS Error/Event Return Format

This section describes in detail the return value retrieved by an RTAS call to either the *event-scan* or *check-exception* function.

The return value consists of a fixed part and an optional Extended Error Report, described in the next section, which contains full details of the error. The fixed part is intended to allow reporting the most common problems in a simple way, which makes error detection and recovery simple for OSs that want to implement a very simple error handling strategy. At the same time, the mechanism is capable of providing full disclosure of the error syndrome information for OSs which have a more complete error handling strategy.

RTAS can return at most one return code per invocation. If multiple conditions exist, RTAS returns them in descending order of severity on successive calls.

10.3.2.1 Reporting and Recovery Philosophy, and Description of Fields

All firmware implementations use a common error and event reporting scheme, as described in detail below. It is not required that error recovery be present in firmware implementations, nor is it required that a high degree of error recovery or survival be undertaken by OSs. If such behavior is desired, then specific platform-dependent handlers can be loaded into the OS. However, this section defines return result codes and a philosophy which can be used if aggressive error handling is implemented in firmware. This section describes the fields of the Error Report format, and the philosophy which should be applied in generating return values from firmware or interpreting such return codes in an OS.

In general, an OS would look at the Disposition field first to see if an error has been corrected already by firmware. If not corrected to the OS's satisfaction, the OS would examine the Severity field. Based on that value, and optionally on any information it can use from the Type and other fields, the OS will make a determination of whether to continue or to halt operations. In either case, it may choose to log information regarding the error, using the remaining fields and optional Extended Error Log.

The following sections describe the field values in Table 137, "RTAS Event Return Format (Fixed Part)," on page 292.

10.3.2.1.1 Version

This field is used only to distinguish among present and potential future formats for the remainder of the error report. This value will be incremented if extensions are made to the format described here. The primary function of this field is for future OSs to identify whether an error report may contain some (unknown at present) feature that was added after the initial version of this specification.

10.3.2.1.2 Severity

This field represents the value judgment of firmware of how serious the problem being reported should be considered by the OS.

Errors which are believed to represent a permanent hardware failure affecting the entire system are considered "FATAL." OSs would not attempt to continue normal operation after receiving notice of such an error. OSs may not even be able to perform an orderly shutdown in the presence of a Fatal error, though they may make a policy decision to try.

Less serious errors, but still causing a loss of data or state, are considered "ERRORs." In general, continuing after such an error is questionable, since details of what has failed may not be available, or if available, may not map nicely onto any ongoing activity with which the OS can associate it. However, OSs may make a policy decision (for example, based on the error Type, the Initiator, or the Target) to continue operation after an Error.

There are some types of errors, such as parity errors in memory or a parity error on a transfer between CPU and memory, which occur synchronously with the current process execution context. Such errors are sometimes fatal only to the current thread of execution; that is, they affect only the current CPU state and possibly that of any memory locations being currently referenced. If that context of execution is not essential to the system's operation (for example, if an error trap mechanism is available in the OS and can be triggered to recover the OS to a known state), recovery and continuation may be possible. Or at least, since the memory of the machine is in an undamaged state, the system may be able to be brought down in an orderly fashion. Such errors are reported as having Severity "ERROR_SYNC". It is OS dependent whether recovery is possible after such an error, or whether the OS will treat it as a fatal problem.

The "WARNING" return value indicates that a non-state-losing error, either fully recovered by firmware or not needing recovery, has occurred. No OS action is required, and full operation is expected to continue unhindered by the error. Examples include corrected ECC errors or bus transfer failures which were re-tried successfully.

The "EVENT" return value is the mechanism firmware uses to communicate event information to the OS. The event may have been detected by polling using *event-scan* or on the occurrence of an interrupt by calling *check-exception*. In either case, the Error Return value indicates the event which has occurred in the Type field. See the Type description below for a description of specific events and their expected handling.

The "NO_ERROR" return value indicates that no error was present. In this case, the remainder of the Error Return fields are not valid and should not be referenced.

10.3.2.1.3 RTAS Disposition

An aggressive firmware implementation may choose to attempt recovery for some classes of error so an OS can continue operation in the face of recoverable errors. If firmware detects an error for which it has recovery code, it attempts such action before it returns a value to the OS (that is, the mechanisms are linked in RTAS and cannot be separately accessed). Note that Disposition is nearly independent from Severity. Severity says how serious an error was, and Dispo-

sition says, regardless of severity, whether or not the OS has to even look at it. In general, an OS will first examine Disposition, then Severity.

A return value of “FULLY RECOVERED” means that RTAS was able to completely recover the machine state after the error, and OS operation can continue unhindered. The severity of the problem in this case is irrelevant, though for consistency a “FATAL” error can never be “FULLY RECOVERED.”

A return value of “LIMITED RECOVERY” means that RTAS was able to recover the state of the machine, but that some feature of the machine has been disabled or lost (for example, error checking), or performance may suffer (for example, a failing cache has been disabled). The RTAS implementation may return further information in the extended error log format regarding what action was done or what corrective action failed. In general, a conservative OS will treat this return the same as “NOT RECOVERED,” and initiate shutdown. A less conservative OS may choose to let the user decide whether to continue or to shut down.

A value of “NOT RECOVERED” indicates that the RTAS either did not attempt recovery, or that it attempted recovery but was unsuccessful.

10.3.2.1.4 Optional Part Presence

This is a single flag, valid only if the 32-bit Error Return value is located in memory, which indicates whether or not an Extended Error Log Length field and the Extended Error Log follows it in memory. It will be set on an in-memory return result from RTAS if and only if the RTAS call indicated sufficient space to return the Extended Error Log, and the RTAS implementation supports the Extended Error Log.

10.3.2.1.5 Initiator

This field indicates, to the best ability of RTAS to determine it, the initiator of a failed transaction. (Note that in the “Initiator” field of Table 135, “Error Indications for System Operations,” on page 285, the value “I/O” indicates one of the defined I/O buses or IOAs. This field contains finer-grained details of which type of I/O bus failed, if known, and “UNKNOWN” if RTAS cannot tell.)

In many of the newer LoPAPR platforms, the platform error notification and handling flow is asynchronous to the OS and software execution flow, therefore the context of Initiator is not applicable to the platform firmware. In those cases, the value of “(0) Unknown or Not Applicable” is used for Initiator. In logs created with Version 6 or later, more detailed information about the error is provided in the Platform Event log format.

10.3.2.1.6 Target

If RTAS can determine it, this field indicates the target of a failed transaction.

In many of the newer LoPAPR platforms, the platform error notification and handling flow is asynchronous to the OS and software execution flow, therefore the context of Target is not applicable to the platform firmware. In those cases, the value of “(0) Unknown or Not Applicable” is used for Target. In logs created with Version 6 or later, more detailed information about the error are provided in the Platform Event log format.

10.3.2.1.7 Type

This field identifies the general type of the error or event. In some cases (for example, INTERN_DEV_FAIL), multiple possible events are grouped together under a common return value. In such cases, platform-aware software may use the Extended Error Log to distinguish them. Non-platform-aware software will generally treat all errors of a given type the same, so it generally will not need to access the Extended Error Log information.

In the table, the EPOW values are associated with a Severity of EVENT. All other values will be associated with Severity values of FATAL, ERROR, ERROR_SYNC, or WARNING, and may or may not be corrected by RTAS.

EPOW is an event type which indicates the potential loss of power or environmental conditions outside the limits of safe operation of the platform. See “Environmental and Power Warnings” on page 287 for more information.

The “Platform Error (224)”, introduced for Version 6, generalizes that the error is identified by the platform and the specific details are encoded in the Platform Event log format itself.

The “ibm,io-events (225)” defines a set of event notifications which requires special handling by the OS. For this type of event notification, the Platform Event Log contains the “IO Events” section which identifies additional details associated with the event.

The “Platform information event (226)” indicates the return log should be logged as “Information Log”. These logs indicate key platform events and can be used for reference purposes.

The “Resource deallocation event (227)” indicates an event notification to the OS that a specific hardware resource has experienced recurring recoverable errors with a trend toward unrecoverable. The OS should take action to deallocate the resource from usage to prevent unrecoverable errors. For these types of event notification, the Platform Event Log contains the “Logical Resource Identification” section which identifies the “Logical Entity” by Resource Type and Resource ID, associated with the event.

The “Dump notification event (228)” indicates that a dump file is present in the platform and is available for retrieval by the OS. For this type of event notification, the Platform Event Log contains the “Dump Locator” section which contains additional event specific information.

Additional Type values will be added in future revisions of the specification. If an OS does not recognize a particular event type, it can examine the severity first, and then choose to ignore the event if it is not serious.

10.3.2.1.8 Extended Event Log Length / Change Scope

This optional 32-bit field is present in memory following the 32-bit Event Return value if the Optional Part Presence flag is “PRESENT”, and it indicates the length in bytes of the Extended Event Log information which immediately follows it in memory. The length does not include this field or the Event Return field, so it may be zero. The field is also present for a resource change “Hot Plug” event, such as a PRRN event, and then represents the scope of a resource change.

10.3.2.1.9 RTAS Event Return Format Fixed Part

The summary portion of the error return is designed to fit into a single 32-bit integer. When used as a data return format in memory, an optional Length field and Extended Error Log data may follow the summary. The fixed part contains a “presence” flag which identifies whether an extended report is present.

In the table below, the location of each field within the integer is included in parentheses after its name. Numerical field values are indicated in decimal unless noted otherwise.

Table 137. RTAS Event Return Format (Fixed Part)

Bit Field Name (bit number(s))	Description, Values (Described in Section 10.3.2.1, “Reporting and Recovery Philosophy, and Description of Fields,” on page 289)
Version (0:7)	A distinct value used to identify the architectural version of message.
Severity (8:10)	Severity level of error/event being reported: ALREADY_REPORTED (6) FATAL (5) ERROR (4) ERROR_SYNC (3) WARNING (2) EVENT (1) NO_ERROR (0) reserved for future use (7)

Table 137. RTAS Event Return Format (Fixed Part) ***(Continued)***

Bit Field Name (bit number(s))	Description, Values (Described in Section 10.3.2.1, "Reporting and Recovery Philosophy, and Description of Fields," on page 289)
RTAS Disposition (11:12)	Degree of recovery which RTAS has performed prior to return after an error (value is FULLY_RECOVERED if no error is being reported): FULLY_RECOVERED(0) Note: Cannot be used when Severity is "FATAL". LIMITED_RECOVERY(1) NOT_RECOVERED(2) reserved for future use (3)
Optional_Part_Presence (13)	Indicates if an Extended Error Log Length and Extended Error Log follows this 32-bit quantity in memory: PRESENT (1): The optional Extended Error Log is present. NOT_PRESENT (0): The optional Extended Error Log is not present.
Reserved (14:15)	Reserved for future use (0:3)
Initiator (16:19)	Abstract entity that initiated the event or the failed operation: UNKNOWN (0): Unknown or Not Applicable CPU (1): A CPU failure (in an MP system, the specific CPU is not differentiated here) PCI (2): PCI host bridge or PCI IOA Reserved -- do not reuse (3) MEMORY (4): Memory subsystem, including any caches Reserved -- do not reuse (5) HOT PLUG (6) Reserved for future use (7-15)
Target (20:23)	Abstract entity that was apparent target of failed operation (UNKNOWN if Not Applicable): Same values as Initiator field

Table 137. RTAS Event Return Format (Fixed Part) ***(Continued)***

Bit Field Name (bit number(s))	Description, Values (Described in Section 10.3.2.1, "Reporting and Recovery Philosophy, and Description of Fields," on page 289)
Type (24:31)	General event or error type being reported: Internal Errors: RETRY (1): too many tries failed, and a retry count expired TCE_ERR (2): range or access type error in an access through a TCE INTERN_DEV_FAIL (3): some RTAS-abstracted device has failed (for example, TOD clock) TIMEOUT (4): intended target did not respond before a time-out occurred DATA_PARITY (5): Parity error on data ADDR_PARITY(6): Parity error on address CACHE_PARITY (7): Parity error on external cache ADDR_INVALID(8): access to reserved or undefined address, or access of an unacceptable type for an address ECC_UNCORR (9): uncorrectable ECC error ECC_CORR (10): corrected ECC error RESERVED (11-63): Reserved for future use Environmental and Power Warnings: EPOW(64): See Extended Error Log for sensor value RESERVED (65-95): Reserved for future use Reserved -- do not reuse (96-159) Platform Resource Reassignment (160) -- includes Change Scope in bits 32-63 Reserved for future use (through-223) Platform Error (224) (for Version 6 or later) ibm,io-events (225) (for Version 6 or later) Platform information event (226) (for Version 6 or later) Resource deallocation event (227) (for Version 6 or later) Dump notification event (228) (for Version 6 or later) Vendor-specific events(229-255): Non-architected Other (0): none of the above
Extended Event Log Length / Change Scope (32:63)	Length in bytes of Extended Event Log information which follows (see 10.3.2.2, "Version 6 Extensions of Event Log Format," on page 294) OR the scope parameter to be input the *ibm,update-nodes* RTAS to retrieve the nodes that were changed by selected "Hot Plug" events.

Typically, most OSs care about, and have handlers for, only a few specific errors. Since coding of an error is unique in the above scheme, an OS can check for specific errors, then if nothing matches exactly, look at more generic parts of the error message. This permits generic error message generation for the user, providing the basic information that RTAS delivered to the OS. Platforms may provide more complete error diagnosis and reporting in RTAS, combined with off-line diagnostics which take advantage of the information reported from previous failures.

10.3.2.2 Version 6 Extensions of Event Log Format

10.3.2.2.1 RTAS General Extended Event Log Format, Version 6

The following section defines new extensions to the event log format which are identified by a Version number 0x06 in the first byte in the returned buffer (byte 0 of the fixed-part information). The following tables define extended error log formats for Version 6, by which the RTAS can optionally return detailed information to the software about a hardware error condition. Other versions will be defined in following sections of this chapter. This format is also intended to be usable as residual error log data in NVRAM, so that the OS could alternatively retrieve error data after an error event which caused a reboot.

Platforms indicate the maximum length of the error log buffer in the **"rtas-error-log-max"** RTAS property in the OF device tree, so that the OS can allocate a buffer large enough to hold the extended error log data when calling the RTAS *event-scan* or *check-exception* functions. If the allocated buffer is not large enough to hold all the error log data, the data is truncated to the size of the buffer.

Requirement R1–10.3.2.2.1–1 and Table 138, "RTAS General Extended Event Log Format, Version 6," on page 296 require that four bytes of the vendor-specific format contain a unique identifier for the company that has defined the format. The description of the "name" string in *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2] provides alternatives for defining this identifier. Examples of these unique identifiers include stock ticker labels and Organizationally Unique Identifiers (OUIs). Since the different options in IEEE 1275 provide different guarantees of uniqueness and different identifier lengths, the company should use its best judgement in selecting a unique identifier that fits the four character field. The length of this field is limited to 4 bytes to conserve available log data space. As an example, if Allied Information Monitoring (a fictional name for the purposes of this example) were to create a vendor-specific log format 12, then bytes 12-15 of such a log may contain "AIM<NULL>".

This identifier is intended to apply to the company that defines the specific format, and may be used by other companies that wish to be compatible with that format. For example, if another company wanted to take advantage of existing support in one of the OSs by using an AIM-specific error log format for logs generated on their own platform, their log would have to contain an identifier of "AIM<NULL>".

R1–10.3.2.2.1–1. Platforms which support Version 6 of the Extended Event Log Format must do so by including a 0x06 value in the first byte of the RTAS Event Return Format (Fixed Part) and using the formats described in Section 10.3.2.2, "Version 6 Extensions of Event Log Format," on page 294 (and all subsections under that section).

Software Implementation Note: OSs running on platforms which support Version 6 of the Extended Event Log Format must ensure that the length parameter passed in the *event-scan* RTAS call be at least 2 KB.

R1–10.3.2.2.1–2. If the length parameter on the RTAS *event-scan* call for returning data using Version 6 of the Extended Event Log Format is insufficient to return all the data the platform would otherwise make available, the platform must truncate the data by eliminating optional sections entirely rather than truncating a section.

R1–10.3.2.2.1–3. All event logs returning a Version 6 Platform Event Log format must include the Main-A and Main-B Sections. Other sections are optional depending on the specific event type as specified in Requirement R1–10.3.2.2.1–4.

R1–10.3.2.2.1–4. The following sections must be provided as indicated:

a. For the Platform error Type, the Primary Service Reference Code (SRC) section must be provided.

b. For the ibm,io-events Type, the IO Events section must be provided.

c. For the Resource deallocation event Type, the Logical Resource Identification section must be provided.

d. For the Dump notification event Type, the Dump Locator section must be provided.

e. For the EPOW Type, the EPOW section must be provided.

Software Implementation Note: All fields in the Platform Event Log marked “Platform specific information” or “Other platform specific information sections” contain information reserved for platform or platform Service Application use only. That information is not defined in this document. Information in these fields should be ignored by the OS.

Software Implementation and Architecture Note: All fields currently marked “Reserved” are set to zero by RTAS and are ignored by the OS. The reserved values in the defined fields in the Platform Event Log may be defined in the future in this architecture document for platform specific usage without change to this architecture.

Table 138. RTAS General Extended Event Log Format, Version 6

Byte	Bit	Description
0	0	1 = Log Valid
	1	1 = Unrecoverable Error
	2	1 = Recoverable (correctable or successfully retried) Error
	3	1 = Unrecoverable Error, Bypassed - Degraded operation (e.g. CPU/memory taken off-line, bad cache bypassed, etc.)
	4	1 = Predictive Error - Error is recoverable, but indicates a trend toward unrecoverable failure (e.g. correctable ECC error threshold, etc.)
	5	1 = “New” Log (always 1 for data returned from RTAS)
	6	1 = Big-Endian
	7	Reserved
1	0:7	Reserved
2	0	Set to 1 - (Indicating log is in PowerPC format)
	1:3	Reserved
	4:7	Log format indicator, defined format used for byte 12-2047: 0-13, 15 Reserved 14: Platform Event Log
3	0:7	Reserved
4-11		Reserved
12-15		Company identifier of the company that has defined the format for this vendor specific log type.
16-2047		Detail vendor specific log data. If byte 2, bits 4:7, above, are a value of 14 (Platform Event Log) and bytes 12-16 are “IBM ”, then see Section 10.3.2.2.2, “Platform Event Log Format, Version 6,” on page 296 for the content of this field.

10.3.2.2.2 Platform Event Log Format, Version 6

This format is used when byte 2, bits 4:7, of the RTAS General Extended Event Log Version 6 are a value of 14 (Platform Event Log).

Table 139. Overview of Platform Event Log Format, Version 6

Byte	Length in Bytes	Description
12-15	4	Contains ASCII characters “IBM<NULL>”.
16-63	48	Main-A section (ID = 'PH'). Required section. See Section 10.3.2.2.3, “Platform Event Log Format, Main-A Section,” on page 297 for the format.
64-87	24	Main-B section (ID = 'UH'). Required, always follow Main-A section. See Section 10.3.2.2.4, “Platform Event Log Format, Main-B Section,” on page 298 for the format.
88-103	16	Logical Resource Identification section (ID = 'LR'). Optional, present only for Resource deallocation event notification. If present, this section always follows Main-B section. See Section 10.3.2.2.5, “Platform Event Log Format, Logical Resource Identification section,” on page 303 for the format.
104-	80+ optional FRU call out sub-section	Primary SRC section (ID = 'PS'). Required for “Platform Error” event type, optional for other event types. If present, this section always follows Main-B section. See Section 10.3.2.2.6, “Platform Event Log Format, Primary SRC Section,” on page 304 for the format.
	64	Dump Locator section (ID = 'DH') Optional, present only for dump event notification. If present, this section follows Main-B or Primary SRC section. See Section 10.3.2.2.7, “Platform Event Log Format, Dump Locator Section,” on page 307 for the format.
	20	EPOW section (ID = 'EP'). Optional, present only for “EPOW” interrupt event notification. If present, this section follows Main-B section. See Section 10.3.2.2.8, “Platform Event Log Format, EPOW Section,” on page 308 for the format.
	Variable	IO Events section (ID = 'IE'). Optional, present only for “ibm,io-events” interrupt event notification. If present, this section follows Main-B section. See Section 10.3.2.2.9, “Platform Event Log Format, IO Events Section,” on page 309 for the format.
	28	Failing Enclosure MTMS section (ID = 'MT'). Required for errors only. If present, this section follows Main-B section or Primary SRC. See Section 10.3.2.2.10, “Platform Event Log Format, Failing Enclosure MTMS,” on page 310 for the format.
	28	Impacted partition description section (ID = 'LP"). Required for errors only. If present, this section follows Main-B section or Primary SRC
	40	Machine Check Interrupt section (ID = 'MC'). Optional for “Platform Error” event types with ERROR_SYNC severity caused by a machine check interrupt. If present, this section follows the Main-B. See Section 10.3.2.2.12, “Platform Event Log Format, Failing Memory Address,” on page 311.
...- 2047	Variable	Other platform specific information sections. Optional.

10.3.2.2.3 Platform Event Log Format, Main-A Section

Table 140. Platform Event Log Format, Version 6, Main-A Section

Offset	Length in Bytes	Description
0x00	2	Section ID: A two-ASCII character field which uniquely identifies the type of section. value = 'PH'
0x02	2	Section length: Length in bytes of the section, including the section ID. value = 48
0x04	1	Section Version

Table 140. Platform Event Log Format, Version 6, Main-A Section *(Continued)*

0x05	1	Section sub-type
0x06	2	Creator Component ID
0x08	4	Log creation date in BCD format: YYYYMMDD, where YYYY = year, MM = month 01 - 12, DD = day 01 - 31.
0x0C	4	Log creation time in BCD format: HHMMSS00, where HH = hour 00 - 23, MM = minutes 00 - 59, SS = s econds 00 -5 9, 00 = hundredth of seconds 00 - 99.
0x10	8	Platform specific information
0x18	1	Creator ID -- subsystem creating the log entry represented as a single ASCII character 'E' = Service Processor 'H' = Hypervisor, 'W' = Power Control 'L' = Partition Firmware
0x19	2	Reserved
0x1B	1	Section count -- number of sections comprising log entry, including this section
0x1C	4	Reserved
0x20	8	Platform specific information
0x28	4	Platform Log ID (PLID) Unique identifier for a single event. Note that it is possible for multiple log entries to be made for a single error/event. The entries are linked to the same event by using the same PLID.
0x2C	4	Platform specific information

10.3.2.2.4 Platform Event Log Format, Main-B Section

Table 141. Platform Event Log Format, Version 6, Main-B Section

Byte	Length in Bytes	Description
0x00	2	Section ID: A two-ASCII character field which uniquely identifies the type of section. value = 'UH'
0x02	2	Section length: Length in bytes of the section, including the section ID. value = 24
0x04	1	Section Version
0x05	1	Section subtype
0x06	2	Creator Component ID

Table 141. Platform Event Log Format, Version 6, Main-B Section ***(Continued)***

0x08	1	Subsystem ID: For error events, this is the failing subsystem. For non-error events, this is the subsystem associated with the event. 0x10 - 0x1F = Processor subsystem including internal cache 0x20 - 0x2F = Memory subsystem including external cache 0x30 - 0x3F = I/O subsystem (hub, bridge, bus) 0x40 - 0x4F = I/O adapter, device and peripheral 0x50 - 0x5F = CEC hardware 0x60 - 0x6F = Power/Cooling subsystem 0x70 - 0x79 = Others subsystem 0x7A - 0x7F = Surveillance Error 0x80 - 0x8F = Platform Firmware 0x90 - 0x9F = Software 0xA0 - 0xAF = External environment 0xB0 - 0xFF = Reserved
0x09	1	Platform specific information
0x0A	1	Event/Error Severity (see additional description following the table) 0x00 = Informational or non- error Event. This field must be 0x00 for non-error event. Use Event Sub-type field to specify unique event. 0x1X = Recovered Error 0x10 = Recovered Error, general 0x14 = Recovered Error, spare capacity utilized 0x15 = Recovered Error, loss of entitled capacity 0x2X = Predictive Error 0x20 = Predictive Error, general 0x21 = Predictive Error, degraded performance 0x22 = Predictive Error, fault may be corrected after platform re-boot 0x23 = Predictive Error, fault may be corrected after boot, degraded performance 0x24 = Predictive Error, loss of redundancy 0x4X = Unrecoverable Error 0x40 = Unrecoverable Error, general 0x41 = Unrecoverable Error, bypassed with degraded performance 0x44 = Unrecoverable Error, bypassed with loss of redundancy 0x45 = Unrecoverable Error, bypassed with loss of redundancy and performance 0x48 = Unrecoverable Error, bypassed with loss of function 0x6X = Error on diagnostic test 0x60 = Error on diagnostic test, general 0x61 = Error on diagnostic test, resource may produce incorrect results All other values = reserved
0x0B	1	Event Sub-Type (primarily used when Event Severity = 0x00, see additional description following the table) 0x00 = not applicable. 0x01 = Miscellaneous, Information Only 0x08 = Dump Notification (Dump may also be reported on Error event) 0x10 = Previously reported error has been corrected by system 0x20 = System resources manually deconfigured by user 0x21 = System resources deconfigured by system due to prior error event 0x22 = Resource deallocation event notification 0x30 = Customer environmental problem has returned to normal (e.g. input power restored, ambient temperature back within limits) 0x40 = Concurrent Maintenance Event 0x60 = Capacity Upgrade Event 0x70 = Resource Sparing Event 0x80 = Dynamic Reconfiguration Event (generated by RTAS) 0xD0 = Normal system/platform shutdown or powered off 0xE0 = Platform powered off by user without normal shutdown (abnormal power off) All other values = reserved
0x0C	4	Platform specific information
0x10	2	Reserved

Table 141. Platform Event Log Format, Version 6, Main-B Section *(Continued)*

0x12	2	Error Action Flags (see additional description following the table) bit 0 (0x8000) = 1, Service Action (customer notification) Required bit 1 (0x4000) = 1, Hidden Error - exclusive with SA Required (bit 0) bit 2 (0x2000) = 1, Report Externally (send to HMC and hypervisor) bit 3 (0x1000) = 1, Don't report to hypervisor (only report to HMC) (only meaningful when (bit 2) Report Externally is set) bit 4 (0x0800) = 1, Call Home Required (only valid if (bit 0) SA Required is set) bit 5 (0x0400) = 1, Error Isolation Incomplete. Further analysis required. bit 6 (0x0200) = 1, Deprecated. bit 7 (0x0100) = 1, Reserved bit 8, 9 = Platform specific information bit 10-15 = Reserved
0x14	4	Reserved

10.3.2.2.4.1 Error/Event Severity

This field indicates the severity of the error event and the impact of the error to the platform (if applicable).

Non-error or Informational Event: This value indicates an event that is a non-error event. Informational or user action event log entries must use this value. The Event Type field provides additional event information.

Recovered Error, general: This value indicates an error event that has been automatically recovered or corrected by the platform hardware and/or firmware, e.g. ECC, internal spare or redundancy, cache line delete, boot time array repair, etc. No service action is required for this type of error. In general, when this value is used, the Error Action Flags has the value of "Hidden Error". An event log with this value is used primarily for error thresholding design and code debug or as a record to indicate error frequency or trend.

Recovered Error, spare capacity utilized: This value indicates that an error on a resource has been recovered by utilizing another resource not currently assigned for use (spare). The failing component is to be considered permanently in an error state. For example, a faulty instruction on one processor may be checkpointed and loaded into a spare processor, continuing the operations of the faulty one. In this case the failing component is considered permanently in an error state.

Recovered Error, loss of entitled capacity: This value indicates that an error on a resource has been recovered by utilizing another resource already in use by the system. The failing component is to be considered permanently in an error state. This results in a loss of capacity in the partition that receives the error. For example, a processor already in use may take over the operations of a faulty one. Loss of the faulty processor in the system then results in less capacity being available to the partition receiving the error event. Typically this event would have an event sub-type of "Resource deallocation event notification" and the revised amount of entitled capacity would be found in the Logical Resource Identification Section, Entitled Capacity field.

Predictive Error, general: This value indicates an event that has been automatically recovered or corrected by the platform hardware and/or firmware. However, the frequency of the errors indicates a trend toward (or potential) platform unrecoverable error. A deferred service or repair action is required. The automatic platform recovery actions have no impact to system performance (e.g. ECC, CRC, etc.), or the impact is unknown.

Predictive Error, degraded performance: This value indicates an error event that has been automatically recovered or corrected by the platform hardware and/or firmware. However, the frequency of the errors (i.e. over threshold) indicates a trend toward (or potential) platform unrecoverable error. A deferred service or repair action is required. The automatic platform recovery actions are impacting/degrading system performance.

Predictive Error, fault may be corrected after platform re-boot: This value indicates an error event that has been automatically recovered or corrected by the platform hardware and/or firmware. However, the frequency of the errors (i.e. over threshold) indicates a trend toward (or potential) platform unrecoverable error. A deferred service or repair action is required. The hardware fault may be corrected after platform re-boot as part of the repair action. If the fault

cannot be corrected after re-boot, then a part replacement is required. The automatic platform recovery actions have no impact to system performance (e.g. ECC, CRC, etc.), or the impact is unknown.

Predictive Error, fault may be corrected after platform re-boot, degraded performance: This value indicates an error event that has been automatically recovered or corrected by the platform hardware and/or firmware. However, the frequency of the errors (i.e. over threshold) indicates a trend toward (or potential) platform unrecoverable error. A deferred service or repair action is required. The hardware fault may be corrected after platform re-boot as part of the repair action. If the fault cannot be corrected after re-boot, then a part replacement is required. The automatic platform recovery actions are impacting/degrading the system performance.

Predictive Error, loss of redundancy: This value indicates an error event that has been automatically recovered or corrected by the platform hardware and/or firmware. However, the frequency of the errors (i.e. over threshold) caused a loss in hardware redundancy. Future error in this subsystem may causes platform unrecoverable error. A deferred service or repair action is required to restore redundancy. The loss of redundancy may or may not impact system performance.

Unrecoverable Error, general: This value indicates an error event that is unrecoverable or uncorrectable by the platform hardware and/or firmware. The hardware or platform resource with the error cannot be deconfigured from the system. If the error is intermittent or soft, the platform may be able to re-boot successfully and resume. A service or repair action is required as soon as possible to correct the error.

Unrecoverable Error, bypassed with degraded performance: This value indicates an error event that is unrecoverable or uncorrectable by the platform hardware and/or firmware. However, the hardware or platform resource with the error has been deconfigured from the system. The platform can be IPLed or re-IPLed with the error bypassed. System performance is degraded due to the deconfigured platform resource(s) e.g. processor, cache, memory, etc. A deferred service or repair action is required.

Unrecoverable Error, bypassed with loss of redundancy: This value indicates an error event that is unrecoverable or uncorrectable by the platform hardware and/or firmware. However, the hardware or platform resource with the error can be deconfigured from the system. The platform can be IPLed or re-IPLed with the error bypassed. The deconfigured platform resource(s) resulted in loss of redundancy (e.g. Redundant FSP with static fail-over) with no loss of system performance. A deferred service or repair action is required.

Unrecoverable Error, bypassed with loss of redundancy + performance: This value indicates an error event that is unrecoverable or uncorrectable by the platform hardware and/or firmware. However, the hardware or platform resource with the error can be deconfigured from the system. The platform can be IPLed or re-IPLed with the error bypassed. The deconfigured platform resource(s) resulted in loss of redundancy and system performance. A deferred service or repair action is required.

Unrecoverable Error, bypassed with loss of function: This value indicates an error event that is unrecoverable or uncorrectable by the platform hardware and/or firmware. However, the hardware or platform resource with the error can be deconfigured from the system. The platform can be IPLed or re-IPLed with the error bypassed. The deconfigured platform resource(s) resulted in loss of platform or system function. A deferred service or repair action is required.

Error on diagnostic test, general: This value indicates an error event that is detected during a diagnostic test. Impact to the system is undefined or unknown.

Error on diagnostic test, resource may produce incorrect results: This value indicates an error event that is detected during a diagnostic test. The error may produce incorrect computational results (e.g. processor floating point unit test error).

10.3.2.2.4.2 Event Sub-Type

This field provides additional information on the non-error event type.

Not applicable: This value is used when the event is associated with an error. Error/Event Severity field and SRC section provide additional error information.

Miscellaneous, Information Only: This value is used when the event is “for information only” or the event description doesn't fit into any other defined values in this field.

Dump Notification: This value is used by the hypervisor or partition firmware as a “Dump Notification” event to the OS that a dump file is present in the platform for retrieval by the OS. This value is used by the HMC as a “Dump Notification” event to the Service Application to indicate a dump file is present for transmission to the manufacturer.

Previously reported error has been corrected by system: This value is used by the platform firmware to indicate that the error event that was previously reported has been corrected by the platform. On a subsequent platform boot, this event type is logged to indicate that the array was successfully repaired.

System resources manually deconfigured by user: This value is used by the platform firmware to indicate that a subset of platform resource(s) was/were deconfigured due to user's request (e.g. via platform ASM menu). The deconfigured resource(s) is/are not associated with error detected by the platform. The event is a reminder to the user that the platform is running with partial capacity. Note: The platform provides this user option for platform performance testing purpose.

System resources deconfigured by system due to prior error event: This value is used by the platform firmware to indicate that the platform is IPLed with resource(s) deconfigured due to error detected and reported previously. The event is a reminder to the user that the platform requires service.

Resource deconfiguration notification: This value is used by partition firmware as an “Event Notification” to the OS that a specified resource (e.g. processor, memory page, etc.) currently used by the OS should be deallocated due to predictive error. A Logical Resource Identification section is included in the event log to indicate the Resource Type and ID.

Customer environmental problem has returned to normal: This value is used by the platform firmware to indicate that a customer environmental problem (e.g. utility power, room ambient temperature, etc.) detected and reported previously, has returned to normal.

Concurrent Maintenance: This value is used by the platform firmware to indicate any non-error event associated with concurrent maintenance activity.

Capacity Upgrade Event: This value is used by the platform firmware to indicate any non-error event associated with capacity upgrade activity.

Resource Sparing Event: This value is used by the platform firmware to indicate any non-error event associated with platform resource sparing activity.

Dynamic Reconfiguration Event: This value is used by the partition firmware to indicate any significant but non-error event associated with dynamic reconfiguration activity. Implementation Note: Due to limited platform storage resource, non-error event log associated with a logical partition will be reported to the OS but may not be stored in the platform.

Normal system/platform shutdown or powered off: This value is used by the platform firmware to indicate any non-error event associated with normal system/platform shutdown or powered off activity initiated by the user.

Platform powered off by user without normal shutdown (abnormal powered off): This value is used by the platform firmware to indicate that the platform is abnormally powered off by the user.

10.3.2.2.4.3 Error Action Flags

The following are the definitions of the actions taken for the various Error Action Flags.

Report Externally - This flag instructs the service processor (error logger component) to send the error to the service application (e.g. service focal point(s) or FNM error analyzer). If this flag is set, the SP always sends the error:

- To the “managing HMC(s)” if one (or multiple) exists.

- And to the hypervisor (unless the “Don't report to hypervisor” flag is also set).

Service Action Required - This flag instructs the Service application that some service action is required by either the customer or by the manufacturer’s service personnel. This is equivalent to saying Customer Notification is required. Contrast this flag with the “Call Home Required” flag.

Call Home Required - This flag indicates that the error requires service and a Call Home Operation is to be performed. There are additional policies used in combination with this flag: what subsystem performs the Call Home, what is sent and where it is sent.

Hidden Error - This flag allows errors to be placed in a partition's OS error log, but still remain hidden from the customer. This is a legacy function and the partition firmware for must filter errors marked “Hidden” and not forward these errors marked with this flag to the OS. Note that this flag has no impact on the SP reporting errors to either the HMC or hypervisor or for the hypervisor reporting errors to partitions.

Don't report Error to hypervisor - While a partition is booting and before it is functional (e.g. no OS error logging available), partition errors may be sent through the hypervisor to the Service Processor). These partition errors (and only partition errors) may be marked with this flag to indicate that they need not be sent back to the hypervisor. This is due to the error scope being limited to the failing partition and the hypervisor has already taken the appropriate actions.

Incomplete Information for Error Isolation - Some errors are not contained to a single enclosure and require error isolation from an entity with broader system view / scope.

Software Error - This flag is used by the partition error logger to indicate to the error is most likely to be caused by the software. When both Software Error and Hardware Error flags are set, the error is caused by either software or hardware. The Software Error and Hardware Error flags are used to trigger the manufacturer’s support system to automatically download software or firmware fixes.

Hardware Error - This flag is used by the partition error logger to indicate to the error is most likely to be caused by the hardware. The Software Error and Hardware Error flags are used to trigger the manufacturer’s support system to automatically download software or firmware fixes.

10.3.2.2.5 Platform Event Log Format, Logical Resource Identification section

Table 142. Platform Event Log Format, Version 6, Logical Resource Identification Section

Offset	Length in Bytes	Description
0x00	2	Section ID: A two-ASCII character field which uniquely identifies the type of section. value = 'LR'
0x02	2	Section length: Length in bytes of the section, including the section ID. value = 20
0x04	1	Section Version
0x05	1	Section subtype
0x06	2	Creator Component ID
0x08	1	Resource Type 0x10: Processor 0x11: Shared processor 0x40: Memory page 0x41: Memory LMB All other values = reserved
0x09	1	Reserved

Table 142. Platform Event Log Format, Version 6, Logical Resource Identification Section ***(Continued)***

0x0A	2	Entitled Capacity: Hundredths of a CPU (only used for Resource Type = Shared processor, value = 0x0000 for others)
0x0C	4	Logical CPU ID: for resource type = processor DRC Index, for resource type = memory LMB Memory Logical Address (bit 0-31), for resource type = memory page
0x10	4	Memory Logical Address (bit 32-64)), for resource type = memory page

10.3.2.2.6 Platform Event Log Format, Primary SRC Section

Table 143. Platform Event Log Format, Version 6, Primary SRC Section

Offset	Length in Bytes	Description
0x00	2	Section ID: A two-ASCII character field which uniquely identifies the type of section. value = 'PS'
0x02	2	Section length: Length in bytes of the section, including the section ID. value = 80 + optional FRU call out sub section
0x04	1	Section Version
0x05	1	Section Subtype
0x06	2	Creator component ID
0x08	1	SRC Version
0x09	1	SRC Flags bit 0:6 = Platform specific information bit 7 = 1: Additional/Optional sub-sections present
0x0A	6	Platform specific information
0x10	4	Extended Reference Code hex data word 2 (required)
0x14	4	Extended Reference Code hex data word 3 (optional)
0x18	4	Extended Reference Code hex data word 4 (optional)
0x1C	4	Extended Reference Code hex data word 5 (optional)
0x20	4	Extended Reference Code hex data word 6 (optional)
0x24	4	Extended Reference Code hex data word 7 (optional)
0x28	4	Extended Reference Code hex data word 8 (optional)
0x2C	4	Extended Reference Code hex data word 9 (optional)
0x30	32	Primary Reference Code: 32 byte ASCII character (required)
Additional/Optional Sub section for FRU call out (present only for “Platform Error” event type)		
0x00	1	Sub section ID = C0 for FRU call out
0x01	1	Platform specific information
0x02	2	Length of sub section: expressed in # of words (4 bytes), from Sub section ID field

Table 143. Platform Event Log Format, Version 6, Primary SRC Section ***(Continued)***

	FRU call out structure length	FRU call out 1 (see FRU call out structure format below)
		FRU call out 2 (call out 2-10 are optional)
		...
		FRU call out 10 (maximum)

Table 144. Platform Event Log Format, Version 6, FRU Call-out Structure

Offset	Length in Bytes	Description
0x00	1	Call-out Structure length, in bytes including all fields, including this one.
0x01	1	Call-out Type / Flags bits 0-3: Call-out structure type 0b0010 = this structure bit 4 = 1 FRU Identity (ID) Substructure field included in this FRU Call-out structure bit 5 = 1 Other platform-only use substructure field present following FRU ID substructure bit 6-7 = 0b11: Other platform-only use substructure field present following FRU ID substructure
0x02	1	FRU Replacement or Maintenance Procedure Priority (expressed as an ASCII character, see additional description following the table) 'H' = High priority and mandatory call-out. 'M' = Medium priority. 'A' = Medium priority group A (1st group). 'B' = Medium priority group B (2nd group). 'C' = Medium priority group C (3rd group). 'L' = Low priority.
0x03	1	Length of Location Code field - must be a multiple of 4.
0x04	variable max=80	Location Code NULL terminated ASCII string. May be up to 80 characters including the NULL. Padded with extra NULLs to 4-byte boundary.
FRU Identity Substructure follow:		
0x00	2	Substructure Type (2 ASCII Characters) 'ID' = FRU Identity Substructure
0x02	1	Substructure length (variable, several optional fields - see flags below)
0x03	1	Flags bits 0-3: Failing component Type (see additional description following the table) 0b0000: reserved 0b0001: “normal” hardware FRU 0b0010: code FRU 0b0011: configuration error, configuration procedure required 0b0100: Maintenance Procedure required 0b1001: External FRU 0b1010: External code FRU 0b1011: Tool FRU 0b1100: Symbolic FRU 0b1111: Reserved for expansion all other values reserved bit 4 (0x08) = 0b1: FRU Stocking Part Number supplied (mutually exclusive with bit 6) bit 5 (0x04) = 0b1: CCIN supplied (only valid if bit 4 = 0b1) bit 6 (0x02) = 0b1: Maintenance procedure call out supplied (mutually exclusive with FRU p/n) bit 7 (0x01) = 0b1: FRU Serial Number supplied (only valid if bit 4 = 0b1)

Table 144. Platform Event Log Format, Version 6, FRU Call-out Structure *(Continued)*

0x04	8, if present	FRU Stocking Part Number (VPD FN keyword) or Procedure ID This field is present if Flags bits 4 =0b1 or Flags bits 6 =0b1. It contains a NULL-terminated ASCII character string. If Flags bit 4 = 0b1, this field contains a 7ASCII character part number If Flags bit 6 = 0b1, this field contains a 5 ASCII character procedure ID
0x0C	4, if present	CCIN (VPD CC keyword) (optional, only supplied if Part Number also supplied) This field is present if Flags bit 5 = 0b1. It contains the CCIN of the failing FRU (VPD CC keyword), represented as 4 ASCII characters (not a NULL-terminated string).
	12, if present	FRU Serial Number (VPD SE Keyword) (optional) This field is present if Flags bit 7 = 0b1. It contains the serial number of the failing FRU (VPD SE keyword), represented as a 12 ASCII characters (not a NULL-terminated string).
End of FRU Identify Substructure		
	variable	Other platform used only substructure field

10.3.2.2.6.1 FRU Replacement or Maintenance Procedure Priority

This field defines the service priority of the specific call-out, i.e., replacing the FRU part number or performing the maintenance procedure ID as given in the FRU/Procedure Identity substructure. Here are the priority descriptions:

- **'H'** = High priority and mandatory call-out. Replacing the FRU (or performing the maintenance procedure) is mandatory. If multiple call-outs with 'H' priority are given, all must be replaced or performed as a group.
- **'M'** = Medium priority. Replacing the FRU (or performing maintenance procedure) with 'M' priority **one at a time in the order given** after all call-outs prior to this one, if present, are performed.
- **'A'** = Medium priority group A (1st group). Replacing all the FRUs **with 'A' priority as a group** after all call-outs prior to this group, if present, are performed.
- **'B'** = Medium priority group B (2nd group). Replacing all the FRUs **with 'B' priority as a group** after all call-outs prior to this group, if present, are performed.
- **'C'** = Medium priority group C (3rd group). Replacing all the FRUs **with 'C' priority as a group** after all call-outs prior to this group, if present, are performed.
- **'L'** = Low priority. After performed all the prior call-outs, if present, and problem still persists, replacing the FRU with this priority **one at a time in the order given**.

The list of FRU/Procedure call-outs in the “call-out” subsection of the SRC structure must be in order as defined above, i.e. High, Medium, Low. 'M' has the same medium priority level as 'A', 'B', or 'C' and a call out with 'M' priority can precede or follow 'A', 'B' or 'C'. A group call-out must be contiguous in the list. Within the medium priority level, follow the call-out order in the list A list without High or Medium priority is also valid.

10.3.2.2.6.2 Failing Component Type Description

- **Normal Hardware FRU:** Hardware FRU in the platform which the platform firmware or code can positively identify, and its VPD contains the part number and associated information.
- **Code FRU:** Some layer of platform firmware or OS code is suspected. The procedure ID field provides additional information about which code(s) is/are the potential problem.
- **Configuration error:** The problem may be related to how hardware or code is configured. For example, an adapter is plugged in a slot that cannot support it. The FRU could be a procedure or a symbolic FRU. The reason to use one of these is if the analysis can provide more information to the customer and service provider by giving a location code.

- **Maintenance procedure required:** Further isolation of the problem is required by performing the procedure as identified in the Procedure ID field. Procedures are designed to help to isolate problems and guide the service provider through identifying which FRUs to replace in which order.
- **Symbolic FRU:** Used for a single FRU where the analysis code knows exactly what the part is but there is no part number, or the part number cannot be pulled from VPD, or when there is something special (like a procedure) for acquiring the FRU or working with it. Examples are cables, or FRUs without VPD (so a part number cannot be filled in). The term “Symbolic” simply means “not an actual part number”.
- **External FRU:** A failing part(s) which is/are not in the system, e.g. attached storage sub-system, network hubs/switches, external drives like CD/DVD boxes.
- **External Code:** Code not running in the platform but is the potential source of the error. This could be something like storage subsystem code or even another system in the same cluster.
- **Tool FRU:** This is a special tool that will be required by one of the FRUs in the list. Tools are only added as FRUs when they are not part of the CE tool kit and therefore the repair action could be delayed if the CE did not know to bring it. Examples are Optical Cleaning Kits for fiber channel, and special tools for torque or reach or weight considerations.

10.3.2.2.7 Platform Event Log Format, Dump Locator Section

Table 145. Platform Event Log Format, Version 6, Dump Locator Section

Offset	Length in Bytes	Description
0x00	2	Section ID: A two-ASCII character field which uniquely identifies the type of section. value = 'DH'
0x02	2	Section length: Length in bytes of the section, including the section ID. value = 64
0x04	1	Section Version
0x05	1	Section Sub-Type 0x00 = Log truncated, complete log received by another service entity. 0x01 = FSP Dump 0x02 = Platform System Dump 0x03 = Reserved 0x04 = Power Subsystem Dump 0x05 = Platform Event Log Entry Dump (when distinguishing between dump types, the term “Log Dump” is typically used) 0x06 = Partition-initiated resource dump 0x07 = Platform-initiated resource dump All other values reserved
0x0	2	Creator component ID
0x08	4	Dump ID
0x0C	1	Flags bit 0 (0x80) = 0, Dump sent to partition bit 0 = 1, Dump sent to HMC bit 1 (0x40) = 0, File name in ASCII bit 1 = 1, Dump file name is hex bit 2 (0x20) = 1, Dump size field valid
0x0D	2	Reserved
0x0F	1	Length of OS assigned Dump ID field in bytes, must be multiple of 4. May be 0.

Table 145. Platform Event Log Format, Version 6, Dump Locator Section *(Continued)*

0x10	8	Dump Size
0x18	40	OS-Assigned Dump ID As the flag field indicates, this field may either be an ASCII string or a hex number. When an ASCII string (AIX, Linux, HMC), this is a NULL terminated ASCII string representing the dump file name (leaf name only, does not include path). Field may be up to 40 characters including the NULL.

10.3.2.2.8 Platform Event Log Format, EPOW Section

Table 146. Platform Event Log Format, Version 6, EPOW Section

Offset	Length in Bytes	Description
0x00	2	Section ID: A two-ASCII character field which uniquely identifies the type of section. value = 'EP'
0x02	2	Section length: Length in bytes of the section, including the section ID.
0x04	1	Section Version
0x05	1	Section subtype
0x06	2	Creator Component ID
0x08	1	EPOW Sensor Value (low-order 4 bits contain the action code).
0x09	1	EPOW Event Modifier (low-order 4 bits contain the event modifier value) 0x00 = Not applicable For EPOW sensor value = 3 0x01 = Normal system shutdown with no additional delay 0x02 = Loss of utility power, system is running on UPS/Battery 0x03 = Loss of system critical functions, system should be shutdown 0x04 = Ambient temperature too high All other values = reserved
0x0A	1	Extended Modifier for Section Version 2 and higher For EPOW Sensor Value = 3 0x00 System wide shutdown 0x01 Partition specific shutdown 0x02 – 0xFF Reserved All other situations Reserved = 0x00
0x0B	1	Reserved
0x0C	8	Platform specific reason code

10.3.2.2.9 Platform Event Log Format, IO Events Section

Table 147. Platform Event Log Format, Version 6, IO Events Section

Offset	Length in Bytes	Description
0x00	2	Section ID: A two-ASCII character field which uniquely identifies the type of section. value = 'IE'
0x02	2	Section length: Length in bytes of the section, including the section ID.
0x04	1	Section Version
0x05	1	Section subtype
0x06	2	Creator Component ID
0x08	1	IO-Event Type: 0x01 = Error Detected 0x02 = Error Recovered 0x03 = Event 0x04 = RPC Pass Through All other values = Reserved
0x09	1	Offset 0x10 Field Length: For IO Event Type of RPC Pass Through, this field specifies the length of the data field which begins at offset 0x10, otherwise the value in this field is 0. Must be a multiple of 4 to maintain 4-byte alignment.
0x0A	1	Error/Event Scope: 0x00 = Not Applicable (use for IO-Event type 0x02, 0x03, 0x04) 0x36 = Reserved 0x37 = Reserved 0x38 = PHB 0x39 = Reserved 0x3A = Reserved 0x3B = Reserved 0x51 = Service Processor All other values = Reserved
0x0B	1	I/O-Event Sub-Type: 0x00 = Not Applicable (use for IO-Event type 0x01, 0x02, 0x04) 0x01 = Rebalance request 0x03 = Node online 0x04 = Node off-line 0x05 = platform-dump-max-size change 0x08 = Generic Notification All other values = Reserved
0x0C	4	DRC Index
0x10	0-216	For the RPC Pass Through IO Event Type: RPC data. Variable length data. Must be padded to 4 bytes alignment. For the platform-dump-max-size change I/O-Event Sub-Type: 8 bytes for the new value of the platform-dump-max-size system parameter (specifying the sum (in bytes) of the maximum size of each unique platform dump type that the *ibm,platform-dump* RTAS call could return). For Generic Notification I/O Event Sub-Type: Scoped Data Generic Notification Event Section. Must be padded to 4 bytes alignment.

10.3.2.2.10 Platform Event Log Format, Failing Enclosure MTMS

Table 148. Platform Event Log Format, Version 6, Failing Enclosure MTMS

Offset	Length in Bytes	Description
0x00	2	Section ID: A two-ASCII character field which uniquely identifies the type of section. value = 'MT'
0x02	2	Section length: Length in bytes of the section, including the section ID. value = 28
0x04	1	Section Version
0x05	1	Section subtype
0x06	2	Creator Component ID
0x08	8	Machine Type and Model: 8 ASCII characters, in the form “tttt-mmm”, where tttt = Machine Type and mmm = Model Number
0x10	12	Serial Number: 12 ASCII characters (If less than 12 characters are used, string is left justified (stored in the field starting with the lowest address) and padded with NULLs.)

The Failing Enclosure Machine Type, Model, and Serial Number (MTMS) that is associated with the error is important for service and support.

The source of information for the MTMS fields varies according to the following:

- For CEC errors, it is the CEC enclosure MTMS.
- For errors in I/O enclosures (drawers and towers) that have their own MTMS and are sold as separate MTMS from the CEC, we use the I/O Drawer MTMS.
- For I/O enclosures that were sold as a feature, this section contains the Feature Code and Serial Number of the I/O enclosure. When the Feature Code is used, it is left justified in the Machine Type and Model field.

10.3.2.2.11 Platform Event Log Format, Impacted Partitions

Table 149. Platform Event Log Format, Version 6, Impacted Partitions

Offset	Length	Byte 0	Byte 1	Byte 2	Byte 3
0	8	Section Header			
0x10	4	Primary Partition ID	Length of LP name (must be a multiple of 4)	Target LP Count	
0x14	4	Logical Partition ID			
0x18	variable	Primary Partition (LP) Name Null terminated ASCII string, padded to 4-Byte boundary			
	variable	Target LP 1	Target LP 2 and so on (padded to a 4-Byte boundary)		

This section describes partitions that are impacted by an error. When this section is supplied, the partitions in this list (and only these partitions) are notified of the error.

10.3.2.2.12 Platform Event Log Format, Failing Memory Address

Table 150. Platform Error Event Log Format, Version 6, Failing Memory Address

Offset	Length in Bytes	Description
0x00	2	Section ID: A two-ASCII character field which uniquely identifies the type of section. value = 'MC'
0x02	2	Section length: Length in bytes of the section, including the section ID value = 32
0x04	1	Section Version
0x05	1	Section Subtype
0x06	2	Creator Component ID
0x08	4	FRU ID -- Identifies the FRU on which the machine check interrupt occurred
0x0C	4	Processor ID -- identifies the physical CPU on which the machine check occurred
0x10	1	Type of machine check interrupt 0x00 = Uncorrectable Memory Error (UE) 0x01 = SLB error 0x02 = ERAT Error 0x04 = TLB error 0x05 = D-Cache error 0x07 = I-Cache error
0x11	23	Information specific to machine check interrupt type. This section is binary zeroes if the platform does not provide specific information for the type of interrupt.

Table 151. UE Error Information

Offset	Length in Bytes	Description
0x11	1	Type of UE Bit 0 = 0 Permanent UE. The UE may be cleared with a DCBZ instruction. Bit 0 =1 Transient UE. The UE cannot be cleared with a DCBZ instruction. The contents of the entire logical page are not accessible for this type of UE 64 bit effective address is provided Bit 1 = 0 64 bit effective address is not provided by the log Bit 1 = 1 64 bit effective address is provided by the log. Offset 0x18 provides the effective address if this bit is 1 64 bit logical address is provided Bit 2 = 0 64 bit logical address of logical page is not provided by the log Bit 2 = 1 64 bit logical address of logical page is provided by the log. Offset 0x20 provides the logical address of the page if this bit is 1 Bit3-4 Reserved Bit5-7 Type of UE machine check interrupt. The value of the field is 0b000 for a permanent UE 0b000 = Platform cannot determine the processor unit that detected the error 0b001 = Error detected by instruction fetch unit of the processor 0b010 = Error during page table search for instruction fetch 0b011 = Error detected by load/store unit of the processor 0b100 = Error detected during page table search for load/store type of instruction All other values are reserved.
0x12	6	Reserved
0x18	8	64 bit effective address
0x20	8	64 bit logical address

Table 152. SLB Error Information

Offset	Length in Bytes	Description
0x11	1	64 bit effective address is provided Bit 0 = 0 64 bit effective address not provided by the log Bit 0 =1 64 bit effective address provided by the log. Offset 0x18 provides the effective address if bit 0 is1 Bit1-5 Reserved Bit 6-7 Type of SLB error 0b00 = Parity error in the SLB array or on the access path to the SLB 0b01 = Multiple hit error. There are two or more entries in the SLB that translate the same effective address 0b10 = Multiple hit error or parity error. Platform does not have enough information to disambiguate between the two cases. All other values are reserved.
0x12	6	Reserved
0x18	8	64 bit effective address
0x20	8	Reserved

Table 153. ERAT Error Information

Offset	Length in Bytes	Description
0x11	1	64 bit effective address is provided Bit 0 = 0 64 bit effective address not provided by the log Bit 0 =1 64 bit effective address provided by the log. Offset 0x18 provides the effective address if bit 0 is1 Bit 1-5 Reserved Bit 6-7 Type of ERAT error 0b01 = Parity error in the ERAT array 0b10 = Multiple hit error. There are two or more entries in the ERAT array that translate the same effective address 0b11 = Multiple hit error or parity error in the ERAT array. Platform does not have enough information to disambiguate between the two cases. All other values are reserved.
0x12	6	Reserved
0x18	8	64 bit effective address
0x20	8	Reserved

Table 154. TLB Error Information

Offset	Length in Bytes	Description
0x11	1	64 bit effective address is provided Bit 0 = 0 64 bit effective address not provided by the log Bit 0 =1 64 bit effective address provided by the log. Offset 0x18 provides the effective address if bit 0 is1 Bit 1-5 Reserved Bit 6-7 Type of TLB error 0b01 = Parity error in the TLB array 0b10 = Multiple hit error. There are two or more entries in the TLB that translate the same effective address 0b11= Multiple hit error or parity error in the TLB array. Platform does not have enough information to disambiguate between the two cases. All other values are reserved.
0x12	6	Reserved
0x18	8	64 bit effective address
0x20	8	Reserved

For an error log that has the machine check interrupt section filled out, the platform is not required to provide the date and time stamp in the main-a section. The fields will be binary zeroes if the date and time stamp is not provided.

10.3.3 Location Codes

This document defines an architecture extension for physical location codes. One use of location codes is to append failing location information to error logs returned by the *event-scan* and *check-exception* RTAS services. Refer to Section 12.3, “Hardware Location Codes,” on page 327 for more information on the format and use of location codes. For event logs with Version 6 or later, the location code of FRU call out is contained in the Primary SRC section, FRU call out sub-section of the Platform Event Log format.

10.4 Error Codes

10.4.1 Displaying Codes on the Standard Operator Panels

R1–10.4.1–1. Platform Implementation: Platforms must display firmware progress codes (4 hex digits) on the operator panel display. On 2x16 LCD displays, the progress codes are displayed left-justified on the first line.

R1–10.4.1–2. Platform Implementation: Platforms must display firmware error codes (8 hex digits) on the system console (graphic or tty), and left-justified on the first line of a 2x16 LCD operator panel display (if available).

R1–10.4.1–3. Platform Implementation: When a platform displays firmware error codes, associated location codes must be displayed on the following line on the system console (graphic or tty), and left-justified on the second line of a 2x16 LCD operator panel display (if available).

The following describes in more detail the standard platform usage of operator panel LEDs or LCDs for the display of firmware progress and error codes.

- Progress codes: Progress codes from the system firmware and service processor firmware are 4 hex digits in the range from 0x8000 through 0xFFFF. Codes are displayed in the 4 character positions of a 1x4 LED, or left justified in the first line of a 2x16 LCD. Subsequent progress codes are displayed on top of (overlaying) the previous one. If the system “hangs”, the last displayed progress code is left on the display.
- Error codes: Error codes are 8 hex digits, as defined in Section 10.4.2, “Firmware Error Codes,” on page 314. These codes are displayed by either boot ROM Power On Self Test (POST) or the service processor. If a critical error is detected which prevents a successful boot or results in system halt condition, the error code will be displayed left justified on the first line of a 2x16 LCD. The error code is left on the LCD until the system is reset or powered down. Error codes are not displayed on the operator panel of platforms with only a 4-digit LED. On all platforms, however, POST error codes are displayed on any system console (graphic or tty). For non-critical errors where the system can boot and operate normally or in a degraded mode, the associated error codes are not displayed, but are reported to the OS via the POST error log and the RTAS *event-scan* service.
- Location Codes: Location codes describe the physical location of the most probable failing part associated with an error code. When an error code is displayed on the first line of a 2x16 LCD, the location code, if known, is displayed left justified on the second line. The location code will remain on the LCD along with the error code until the system is reset or powered down. Location codes for POST errors are also displayed on any system console (graphic or tty), on the next line below the error code.

10.4.2 Firmware Error Codes

The error code is an 8-character (4-byte) hexadecimal code produced by firmware to identify the potential failing function or FRU in a system. It consists of 5 source code characters and 3 reason code characters. Individual characters within the error code have specific field definitions, as defined in the following tables.

R1–10.4.2–1. Platform Implementation: To indicate the occurrence of a critical platform error, platforms must display (either on an operator panel or console) an 8-digit hex error code as defined in Table 155, “Service Reference Code (SRC) Field Layout,” on page 315 and Table 156, “Service Reference Code (SRC) Field Descriptions,” on page 315.

Table 155. Service Reference Code (SRC) Field Layout

Source Code					Reason Code		
Byte 0		Byte 1		Byte 2		Byte 3	
S1	S2	S3	S4	S5	R1	R2	R3

Table 156. Service Reference Code (SRC) Field Descriptions

Field	Description
S1	Maintenance Package Source that produced the SRN 0: Reserved 1: Reserved 2: POST, Firmware 3: BIST 4: Service processor, base system controller, etc. 5: Reserved (potentially for use by AIX Diagnostics) 8: Product-Specific Service Guide, MAPs 9: Reserved (potentially for use by the Problem Solving Guide) A-F: Reserved for future extension
S2	Where applicable, use the lower nibble of the *PCI Local Bus Specification* [18] base class code for the IOA definition (see Table 157, “Current PCI Class Code Definition,” on page 316). Only 00 to 0C are currently defined in Revision 2.1, therefore the high nibble is always zero. (There is a potential exposure that the high nibble will be defined in the future, but currently there are 13 base classes defined which include every device class, with 3 remaining characters for future extension by the PCI SIG. Therefore the exposure is in the far future.) For non-PCI devices, use base class 0 to extend the definition (see Table 158, “S2-S3-S4 Definition for Devices/FRUs not Defined in the PCI Specification,” on page 318).
S3-S4	Where applicable, use the *PCI Local Bus Specification* [18] subclass code for IOA definition (see Table 157, “Current PCI Class Code Definition,” on page 316). Also, extend the definition to include non-PCI devices where it is not fully utilized by PCI specification (see Table 158, “S2-S3-S4 Definition for Devices/FRUs not Defined in the PCI Specification,” on page 318).
S5	Unique version of the device/FRU type for a particular product
R1	Device/FRU unique failure reason codes. For POST: assigned by Firmware Developer. For AIX Diagnostics (S1 = 5, not currently supported): 1-7: Use in combination with R2,R3 for diagnostic test failure when maximum isolation was obtained. 8-9: Use in combination with R2,R3 for diagnostic test failure when maximum isolation was NOT obtained. A: Log analysis of POST error log B: Log analysis of machine check or checkstop error log C: Log analysis of AIX device driver error log D: diagnostic detected missing resource E-F: Reserved For others: assigned by respective developers.

Table 156. Service Reference Code (SRC) Field Descriptions

Field	Description
R2-R3	Device/FRU unique failure reason codes. For POST: assigned by Firmware Developer For others: assigned by respective developers.

Table 157. Current PCI Class Code Definition

PCI Base Class (lower nibble) S2	PCI Sub-Class S3-S4	Description
0	Devices that were built before the class code field was defined	
	00	All currently implemented IOAs except VGA-compatible IOAs.
	01	VGA-compatible IOAs.
1	Mass storage controller.	
	00	SCSI bus controller.
	01	IDE controller.
	02	Floppy disk controller.
	03	Intelligent Peripheral Interface (IPI) bus controller.
	04	Redundant Array of Independent Disks (RAID) controller.
	80	Other mass storage controller.
2	Network controller.	
	00	Ethernet controller.
	01	Token Ring controller.
	02	FDDI controller.
	03	ATM controller.
	80	Other Network controller.
3	Display controller.	
	00	VGA-Compatible controller.
	01	Extended Graphics Array (XGA) controller.
	80	Other display controller.
4	Multimedia device	
	00	Video device
	01	Audio device
	80	Other multimedia device

Table 157. Current PCI Class Code Definition ***(Continued)***

PCI Base Class (lower nibble) S2	PCI Sub-Class S3-S4	Description
5	Memory controller.	
	00	RAM
	01	Flash
	80	Other memory controller.
6	Bridge IOAs.	
	00	Host bridge
	01	Reserved
	02	Reserved
	03	Reserved
	04	PCI-to-PCI bridge
	05	Reserved
	06	Reserved
	07	Reserved
	80	Other bridge device.
7	Simple communication controllers.	
	00	Serial controllers.
	01	Parallel port.
	80	Other communication controllers.
8	Generic system peripherals	
	00	PIC
	01	DMA Controller.
	02	System timer
	03	Real-Time Clock (RTC) controller
	80	Other system peripherals
9	Input devices	
	00	Keyboard controller
	01	Digitizer (pen).
	02	Mouse controller
	80	Other input controllers.

Table 157. Current PCI Class Code Definition ***(Continued)***

PCI Base Class (lower nibble) S2	PCI Sub-Class S3-S4	Description
A	Docking stations	
	00	Generic docking station
	80	Other type of docking station
B	Processors	
	20	PA compliant (PowerPC and successors)
	40	Co-processor
C	Serial bus controllers	
	03	Universal Serial Bus (USB)
	04	Fibre Channel

Table 158. S2-S3-S4 Definition for Devices/FRUs not Defined in the PCI Specification

Base Class S2	Sub-Class S3-S4	Description
0	10	AC Power
	11	DC Power
	20	Temperature Related Problem
	21	Fans
	30-3x	Cables
	40-4x	Terminators
	50	Operator panels
	60-6x	Reserved
	70-7x	Reserved
	90-9x	Reserved
	A0	Boot firmware Heartbeat
	B0	O/S Heartbeat
	D0	Unknown device
	E0	Security

Table 158. S2-S3-S4 Definition for Devices/FRUs not Defined in the PCI Specification ***(Continued)***

Base Class S2	Sub-Class S3-S4	Description
1	A0	SCSI Drives (generic)
	B0	IDE Drives
	C0	RAID Drives
	D0	SSA Drives
	E0	Tapes SCSI
	E1	Tapes IDE
	ED	SCSI Changer
	EE	Other SCSI Device
	EF	Diskette drive
	F0	CDROM SCSI
	F1	CDROM IDE
	F2	Read/Write Optical SCSI
	F3	Read/Write Optical IDE
	F4-FF	Reserved for other media devices
5	A0	L2 Cache Controller including integrated SRAM
	A1	L2 Cache SRAM
	A8	NVRAM
	A9	CMOS
	AA	Quartz/EEPROM
	B0-Bx	Memory cards
	Cyy	Memory DIMMs (yy = memory PD bits)
7	A0	I^2C bus
8	A0	Power Management Functions
9	A0-Ax	Keyboards
	B0-Bx	Mouse(s)
	C0-Cx	Dials
	D0	Tablet
	D1-Dx	Reserved for other input devices
B	A0	Service processor

11 The Symmetric Multiprocessor Option

This architecture supports the implementation of symmetric multiprocessor (SMP) systems as an optional feature. This Chapter provides information concerning the design and programming of such systems. For SMP OF binding information, see Section B.7, “Symmetric Multi-Processors (SMP),” on page 729.

SMP systems differ from uniprocessors in a number of ways. These differences are not all covered in this chapter. Other chapters that cover SMP-related topics include:

- Non-processor-related initialization and other requirements: Chapter 2, “System Requirements,” on page 41
- Interrupts: Chapter 6, “Interrupt Controller,” on page 101
- Error handling: Chapter 10, “Error and Event Notification,” on page 281

Many other general characteristics of SMPs—such as interprocessor communication, load/store ordering, and cache coherence—are defined in *Power ISA* [1]. Requirements and recommendations for system organization and time base synchronization are discussed here, along with SMP-specific aspects of the boot process.

SMP platforms require SMP-specific OS support. An OS supporting only uniprocessor platforms may not be usable on an SMP, even when an SMP platform has only a single processor installed; conversely, an SMP-supporting OS may not be usable on a uniprocessor. It is, however, a requirement that uniprocessor OSs be able to run on one-processor SMPs, and that SMP-enabled OSs also run on uniprocessors. See the next section.

11.1 SMP System Organization

This chapter only addresses SMP multiprocessor platforms. This is a computer system in which multiple processors equally share functional and timing access to and control over all other system components, including memory and I/O, as defined in the requirements below. Other multiprocessor organizations (“asymmetric multiprocessors,” “attached processors,” etc.) are not included in this architecture. These might, for example, include systems in which only one processor can perform I/O operations; or in which processors have private memory that is not accessible by other processors.

Requirements R1–11.1–4 through R1–11.1–7, further require that all processors be of (nearly) equal speed, type, cache characteristics, etc. Requirements for optional non-uniform multiprocessor platforms are found in Chapter 15, “Non Uniform Memory Access (NUMA) Option,” on page 505.

R1–11.1–1. OSs that do not explicitly support the SMP option must support SMP-enabled platforms, actively using only one processor.

R1–11.1–2. For the Symmetric Multiprocessor option: SMP OSs must support uniprocessor platforms.

R1–11.1–3. For the Symmetric Multiprocessor option: The extensions defined in Section B.7, “Symmetric Multi-Processors (SMP),” on page 729, and the SMP support section of the RTAS specifications (see Section 7.3.8, “SMP Support,” on page 161) must be implemented.

R1–11.1–4. For the Symmetric Multiprocessor or Power Management option: All processors in the configuration must have equal functional access and "quasi-equal" timing access to all of system memory, including other processors' caches, via cache coherence. "Quasi-equal" means that the time required for processors to access memory is sufficiently close to being equal that all software can ignore the difference without a noticeable negative impact on system performance; and no software is expected to profitably exploit the difference in timing.

R1–11.1–5. For the Symmetric Multiprocessor option: All processors in the configuration must have equal functional and "quasi-equal" timing access to all I/O devices and IOAs. "Quasi-equal" is defined as in Requirement R1–11.1–4, above, with I/O access replacing memory access for this case.

R1–11.1–6. For the Symmetric Multiprocessor option: SMP OSs must at least support SMPs with the same PVR contents and speed. The PVR contents includes both the PVN and the revision number.

R1–11.1–7. For the Symmetric Multiprocessor option: All caches at the same hierarchical level must have the same OF properties.

R1–11.1–8. Hardware for SMPs must provide a means for synchronizing all the time bases of all the processors in the platform, for use by platform firmware. See Section 7.3.8, "SMP Support," on page 161. This is for purposes of clock synchronization at initialization and at times when the processor loses time base state.

R1–11.1–9. The platform must initialize and maintain the synchronization of the time bases and timers of all platform processors such that; for any code sequence "C", run between any two platform processors "A" and "B", where the reading of the time base or timer in processor "A" can be architecturally guaranteed to have happened later in time than the reading of the time base or timer in processor "B", the value of the time base read by processor "A" is greater than or equal to the value of the time base read by processor "B".

Software Implementation Notes:

1. Requirement R1–11.1–1 has implications on the design of uniprocessor OSs, particularly regarding the handling of interrupts. See the sections that follow, particularly Section 11.2.2, "Finding the Processor Configuration," on page 324.

2. While Requirement R1–11.1–6 does not require this, OSs are encouraged to support processors of the same type but different PVR contents as long as their programming models are compatible.

3. Because of performance penalties associated with inter-processor synchronization, the weakest synchronization primitive that produces correct operation should be used. For example, *eieio* can often be used as part of a sequence that unlocks a data structure, rather than the higher-overhead but more general *sync* instruction.

Hardware Implementation Notes:

1. Particularly when used as servers, SMP systems make heavy demands on the I/O and memory subsystems. Therefore, it is strongly recommended that the I/O and memory subsystem of an SMP platform should either be expandable as additional processors are added, or else designed to handle the load of the maximum system configuration.

2. Defining an exact numeric threshold for "quasi-equal" is not feasible because it depends on the application, compiler, subsystem, and OS software that the system is to run. It is highly likely that a wider range of timing differences can be absorbed in I/O access time than in memory access time. An illustrative example that is deliberately far from an upper bound: A 2% timing difference is certainly quasi-equal by this definition. While significantly larger timing differences are undoubtedly also quasi-equal, more conclusive statements must be the province of the OS and other software.

11.2 An SMP Boot Process

Booting an SMP entails considerations not present when booting a uniprocessor. This section indicates those considerations by describing a way in which an SMP system can be booted. It does not pretend to describe "the" way to boot an SMP, since there are a wide variety of ways to do this, depending on engineering choices that can differ from platform to platform. To illustrate the possibilities, several variations on the SMP booting theme will be described after the initial description.

This section concentrates solely on SMP-related issues, and ignores a number of other initialization issues such as hibernation and suspension. See Section 2.1, "System Operation," on page 41 for a discussion of those other issues.

11.2.1 SMP-Safe Boot

The basic booting process described here is called "SMP-Safe" because it tolerates the presence of multiple processors, but does not exploit them. This process proceeds as follows:

1. At power on, one or more finite state machines (FSMs) built into the system hardware initialize each processor independently. FSMs also perform basic initialization of other system elements, such as the memory and interrupt controllers.

2. After the FSM initialization of each processor concludes, it begins execution at a location in ROM that the FSM has specified. This is the start of execution of the system firmware that eventually provides the OF interfaces to the OS.

3. One of the first things that firmware does is establish one of the processors as the *master*: The *master* is a single processor which continues with the rest of the booting process; all the others are placed in a *stopped* state. A processor in this *stopped* state is out of the picture; it does nothing that affects the state of the system and will continue to be in that state until awakened by some outside force, such as an inter-processor interrupt (IPI).[1]

   One way to choose the *master* is to include a special register at a fixed address in the memory controller. That special register has the following properties:

   - The FSM initializing the memory controller sets this register's contents to 0 (zero).
   - The first time that register is read, it returns the value 0 and then sets its own contents to non-zero. This is performed as an atomic operation; if two or more processors attempt to read the register at the same time, exactly one of them will get the 0 and the rest will get a non-zero value.
   - After the first attempt, all attempts to read that register's contents return a non-zero value.

   The *master* is then picked by having all the processors read from that special register. Exactly one of them will receive a 0 and thereby become the *master*.

   Note that the operation of choosing the *master* cannot be done using the PA memory locking instructions, since at this point in the boot process the memory is not initialized. The advantage to using a register in the memory controller is that system bus serialization can be used to automatically provide the required atomicity.

4. The *master* chosen in step 3. then proceeds to do the remainder of the system initialization. This includes, for example, the remainder of Power-On Self Test, initialization of OF, discovery of devices and construction of the OF device tree, loading the OS, starting it, and so on. Since one processor is performing all these functions, and the rest are in a state where they are not affecting anything, code that is at least very close to the uniprocessor code can be used for all of this (but see Section 11.2.2, "Finding the Processor Configuration," on page 324 below).

1.Another characteristic of the *stopped* state, defined by Appendix C, "PA Processor Binding," on page 753, is that the processor remembers nothing of its prior life when placed in a *stopped* state; this distinguishes it from the *idle* state. That isn't strictly necessary for this booting process; *idle* could have been used. However, since the non-*master* processor must be in the *stopped* state when the OS is started, *stopped* might as well be used.

5. The OS begins execution on the single *master* processor. It uses the OF Client Interface Services to start each of the other processors, taking them out of the *stopped* state and setting them loose on the SMP OS code.

This completes the example SMP boot process. Variations are discussed beginning at Section 11.2.3, “SMP-Efficient Boot,” on page 324. Before discussing those variations, an element of the system initialization not discussed above will be covered.

11.2.2 Finding the Processor Configuration

Unlike uniprocessor initialization, SMP initialization must also discover the number and identities of the processors installed in the system. “Identity” means the interrupt address of each processor as seen by the interrupt controller; without that information, a processor cannot reset interrupts directed at it. This identity is determined by board wiring: The processor attached to the “processor 0” wire from the interrupt controller has identity 0. For information about how this identity is used, see Section B.7, “Symmetric Multi-Processors (SMP),” on page 729.

The method used by a platform to identify its processors is dependent upon the platform hardware design and may be based upon service processor information, identification registers, inter-processor interrupts, or other novel techniques.

11.2.3 SMP-Efficient Boot

The booting process as described so far tolerates the existence of multiple processors but does not attempt to exploit them. It is possible that the booting process can be sped up by actively using multiple processors simultaneously. In that case, the pick-a-*master* technique must still be used to perform sufficient initialization that other inter-processor coordination facilities—in-memory locks and IPIs—can be used. Once that is accomplished, normal parallel SMP programming techniques can be used within the initialization process itself.

11.2.4 Use of a Service Processor

A system might contain a service processor that is distinct from the processors that form the SMP. If that service processor has suitably intimate access to and control over each of the SMP processors, it can perform the operations of choosing a *master* and discovering the SMP processor configuration.

12 Product Topology

12.1 VPD and Location Code OF Properties

A set of OF properties is defined to facilitate asset protection and RAS capabilities in LoPAPR systems. The following properties are defined in Section B.6.10.2, “Miscellaneous Node Properties,” on page 717):

- **`"ibm,vpd"`**
- **`"ibm,loc-code"`**

R1–12.1–1. Each instance of a hardware entity (FRU) has a platform unique location code and any node in the OF device tree that describes a part of a hardware entity must include the **`"ibm,loc-code"`** property with a value that represents the location code for that hardware entity.

R1–12.1–2. OF nodes that do not represent an instance of a hardware entity (FRU) do not have a location code and thus these nodes, except for the OF root node, do not include the **`"ibm,loc-code"`** property.

Architecture Note: In general, an OF node that has a unit address corresponds to an instance of a hardware entity and a node that does not have a unit address does not correspond to an instance of a hardware entity. The nodes for caches are one exception to this statement. Note that the OF **`openprom`** node is a system node and thus should have neither the **`"ibm,loc-code"`** property nor the **`"ibm,vpd"`** property.

R1–12.1–3. If a hardware entity contains unique VPD information and the entity corresponds to a node in the OF device tree, then that device tree node must include the **`"ibm,vpd"`** property.

R1–12.1–4. An instance of a hardware entity (FRU) must have one and only one **`"ibm,vpd"`** property element that defines the VPD keywords specified in Requirement R1–12.4.3.1–1.

Platform Implementation Note: In general, an instance of a hardware entity has a single **`"ibm,vpd"`** property element that is in one of the nodes that also contains an **`"ibm,loc-code"`** property element for that entity.

R1–12.1–5. An OF device tree node that includes the **`"ibm,vpd"`** property must also include an **`"ibm,loc-code"`** property, where each property has the same number of elements and the matching elements of both properties define the same instance of a hardware entity (FRU), where matching elements are defined as the nth string of the **`"ibm,loc-code"`** property and the nth set of tags in the **`"ibm,vpd"`** property for all values of n.

R1–12.1–6. VPD data that is not associated with an instance of a hardware entity (FRU) that is described by a single OF device tree node must be reported in an appropriate OF node using the **`"ibm,loc-code"`** and **`"ibm,vpd"`** properties.

Architecture Notes:

- The root node is the appropriate OF node (for Requirement R1–12.1–6) for any instance of a hardware entity that cannot be dynamically reconfigured. Any one and only one of the OF nodes that describe the dynamically reconfigurable entity can contain the properties for the dynamically reconfigurable entity case.
- A dynamically reconfigurable entity may consist of more than one FRU. In this case, the first element of both the **`"ibm,vpd"`** property and the **`"ibm,loc-code"`** property for the dynamically reconfigurable entity must define the VPD and location code for the OF device tree node that includes those properties.

- A FRU can be considered dynamically reconfigurable if the hardware is enabled for dynamic reconfiguration and full platform support might be implemented at a later date.

Platform Implementation Note: If a location does not return VPD, the property would contain a null entry for the VPD.

12.2 System Identification

Appendix B, "LoPAPR Binding," on page 661 provides properties in the "OF Root Node" section called **"system-id"** and **"model"**.

R1–12.2–1. *(Requirement Number Reserved For Compatibility)*

R1–12.2–2. Each system enclosure (generally, a drawer), must have the VPD SE and TM keywords (see Table 160, "LoPAPR VPD Fields," on page 343).

R1–12.2–3. The SE and TM keywords for a master enclosure must be contained in the **"ibm,vpd"** property for the master enclosure which may also contain other keywords. A master enclosure must also have the YK keyword if another enclosure shares the same combined SE and TM keyword values. The YK keyword (see Table 160, "LoPAPR VPD Fields," on page 343) for a secondary enclosure must be contained in the **"ibm,vpd"** property for the secondary enclosure, which may also contain other keywords. Each master enclosure with a YK keyword must have a unique YK value that is the same as the value of the YK keyword for each of the secondary enclosures that share the same combined SE and TM keyword values. An enclosure is not a FRU and thus its VPD must not contain the FN, PN, SN, EC, RM or RL keywords.

R1–12.2–4. The enclosure must also be represented in the corresponding element of the **"ibm,loc-code"** property with the location code of the enclosure for a multi-enclosure platform.

Implementation Note: The location code will be for the full enclosure. The drawer position is the u-units counting from the bottom of the rack. Specific numbering is platform dependent.

R1–12.2–5. If the system contains multiple processor enclosures, firmware must use the VPD SE field from the first or 'marked' processor enclosure (see Table 160, "LoPAPR VPD Fields," on page 343) to construct **"system-id"**.

Implementation Note: What is meant by 'marked' above is that, in a system, the 'first' processor enclosure could become ambiguous. In such a case, a specific processor enclosure could be marked by the firmware or HMC as being the processor enclosure to use for identification purposes.

R1–12.2–6. One or more of the MI, RM, and RL keywords must be present for any firmware specific VPD. (with a value of "UNKNOWN" for unknown values), and must be present based on all of the following:

a. The RL keyword must be provided if the platform does not support the *ibm,update-flash-64-and-reboot* RTAS call.

b. The RM keyword must be provided if the platform supports only a single flash image updated via the *ibm,update-flash-64-and-reboot* service and does not support the *ibm,manage-flash-image* and *ibm,validate-flash-image* RTAS calls.

c. The MI keyword must be provided if the platform supports dual flash images via the *ibm,update-flash-64-and-reboot*, the *ibm,manage-flash-image*, and the *ibm,validate-flash-image* RTAS calls.

d. The ML keyword must be provided if the platform supports dual flash images via the *ibm,update-flash64-and-reboot*, the *ibm,manage-flash-image*, and the i*bm,validate-flash-image* RTAS calls.

R1–12.2–7. System firmware and service processor firmware must have VPD that is separate from the VPD of the FRU that contains the system firmware and service processor firmware and this VPD must have a location

code that is made by adding "/Yn" to the end of the location code for the FRU that contains the firmware where n is a platform dependent instance of the firmware.

R1–12.2–8. The description (large resource type of 0x82) for the system firmware VPD in the **"ibm,vpd"** property must be "System Firmware or Platform Firmware".

R1–12.2–9. System firmware VPD must be found in the **root** node if the system supports only static VPD, otherwise system firmware VPD must be provided via the *ibm,get-vpd* RTAS call.

R1–12.2–10. The description (large resource type of 0x82) for the service processor firmware VPD in the **"ibm,vpd"** property must be "Service Processor Firmware", if the service processor firmware is a separate FRU from the system firmware.

R1–12.2–11. Service processor firmware VPD must be found in the root node if the service processor firmware is a separate FRU from the system firmware.

R1–12.2–12. The VPD SE field (see Table 160, "LoPAPR VPD Fields," on page 343) must be electronically stored in a manner that is not modifiable in a user environment.

R1–12.2–13. There must be a property, **"model"**, under the root node in the format, "<vendor>,xxxx-yyy", where <vendor> is replaced by one to five letters representing the stock symbol of the company (for example, for IBM: "IBM,xxxx-yyy"), and where xxxx-yyy is derived from the VPD TM field (see Table 160, "LoPAPR VPD Fields," on page 343) of the first or 'marked' processor enclosure.

R1–12.2–14. The values of the type/model (TM) and system serial number (SE) (see Table 160, "LoPAPR VPD Fields," on page 343) must remain the same even with service or reconfiguration of the system.

Implementation Note: A change in either TM or SE would indicate a system replacement as opposed to a reconfiguration.

This definition of the **"system-id"** property provides extensibility and ensures that uniqueness can be maintained. The 2 byte Field Type will serve to identify the format of the System Identifier Field which follows.

Hardware Implementation Note: It is recommended that the VPD SE field be held in an area of the machine that is least susceptible to hardware failure. This is to minimize the effect on software licensing when a FRU must be changed. Another useful technique is to socket the part containing the SE field, allowing service personnel to move the old SE to the new FRU.

12.3 Hardware Location Codes

Hardware location codes are used to provide a mapping of logical functions in a platform (or expansion sites for logical functions, such as connectors or ports) to their specific locations within the physical structure of the platform. The description in the following section is intended to define a standard architecture for these location codes, which would be used in three places:

1. These location codes would be included as a property, **"ibm,loc-code"**, under device nodes in the OF device tree, to provide identification of where those device functions are physically located in the platform.

2. To make service and parts replacement easier, hardware location codes of failing components would be appended to the standard 40-byte Extended Error Log templates returned by the RTAS *event-scan* and *check-exception* services.

Software Note: Since the OF device tree currently only defines the core platform devices and IOAs, OS applications such as diagnostics may need to extend these location codes with logical addresses to point to devices such as SCSI (Small Computer System Interface) drives or asynchronous tty (teletypewriter) terminals.

Location Codes are globally unique within a system and are persistent between system reboots and power cycles.

R1–12.3–1. All platforms must implement the Converged Location Codes option.

R1–12.3–2. Location codes must be globally unique within a system, including across multiple CECs of a clustered system, and must be persistent across multiple system reboots and power cycles.

Converged location codes[1] are strings composed of one or more location labels separated by dashes (“-”). Location labels are strings that begin with a location prefix followed by a distinguishing value. For example, the location code for a PCI (Peripheral Component Interconnect) card in a CEC (Central Electronic Complex) might be something like this:

U7879.001.1012345-P2-C3

Valid location codes consist of uppercase characters ('A' - 'Z'), digits (0 - 9), periods ('.'), and dashes ('-'). Characters other than these valid characters will be replaced by pound signs ('#') to protect display and formatting routines and give a consistent indication of a problem in the data. Periods may be used to improve readability of the location code.

The existence of the **“ibm,converged-loc-codes”** property in the **root** node of the device tree indicates that the platform implements the Converged Location Codes option.

R1–12.3–3. For the Converged Location Codes option: The platform must implement the **“ibm,converged-loc-codes”** property in the **root** node of the device tree.

R1–12.3–4. For the Converged Location Codes option: The location code contained in each instance of the property **“ibm,loc-code”** must match the format as described in the sections under Section 12.3.1, “Converged Location Code Labels,” on page 328 and Section 12.3.2, “Converged Location Code Rules,” on page 332.

R1–12.3–5. For the Converged Location Codes option: Extended error logs reported to the OS by *event-scan* or *check-exception* must have the characters “IBM<NULL>” in bytes 12-15, and include the location codes for the failing elements in the format as described in the sections under Section 12.3.1, “Converged Location Code Labels,” on page 328 and Section 12.3.2, “Converged Location Code Rules,” on page 332.

12.3.1 Converged Location Code Labels

This section describes the types of location code labels. See also Section 12.3.2, “Converged Location Code Rules,” on page 332 for the rules for building a location code.

12.3.1.1 Prefix Summary Table

The prefix of a converged location code is as defined in this section.

R1–12.3.1.1–1. For the Converged Location Codes option: The location code prefixes specified in Table 159, “Converged Location Code Prefix Values,” on page 328 must be used in constructing location codes.

Table 159. Converged Location Code Prefix Values

Prefix	Meaning
A	Air handler (for example: blower, fan, motor scroll assembly, motor drive assembly). See Section 12.3.1.4, “Air Handler Location Label,” on page 330.
C	Card Connector (for example, connector for: IOA, Processor Card, Riser Card, Daughter Card, DIMM, Regulator Card, MCM, L3 Cache, Jumper Card, Passthru Interposer (for Processor Fabric when an MCM is not installed), Pluggable Module Chips). See Section 12.3.1.5, “Card Connector Location Label,” on page 330.

1.The term “converged” is historic, based on merging (converging) several previous versions of location codes.

Table 159. Converged Location Code Prefix Values *(Continued)*

Prefix	Meaning
D	Device (for example: Diskette, DASD, Operator Panel, SES (Storage Enclosure Services) Device). See Section 12.3.1.6, “Device Location Label,” on page 330.
E	Electrical (for example: Battery, Power Supply, Charger). See Section 12.3.1.7, “Electrical Location Label,” on page 330.
F	Frame. See Section 12.3.1.15, “Frame Location Label,” on page 331.
L	Logical Path (for example: SCSI Target, IDE Address, ATAPI Address, Fibre Channel LUN, etc.). See Section 12.3.1.10, “Logical Path Label,” on page 331.
M	Mechanical (Plumbing, Valves, Latches). See Section 12.3.1.17, “Mechanical Location Label,” on page 332.
N	Horizontal placement for an empty rack location. See Section 12.3.1.13, “Horizontal Placement Location Label,” on page 331.
P	Planar (for example: Backplane, Board). See Section 12.3.1.3, “Planar Location Label,” on page 329.
R	Resource (identifies a resource that is not a FRU, but needs identification in the error log). See Section 12.3.1.18, “Resource Location Label,” on page 332.
S	SR-IOV adapter virtual function. See Section 12.3.1.16, “Virtual Function Location Label,” on page 331.
T	Port (for example: Port, Connector, Cable Connector, Jack, Interposer). See Section 12.3.1.8, “Port Location Label,” on page 330.
U	Unit (for example: System, CEC, Card Cage, Drawer, Chassis (Unpopulated drawer)). See Section 12.3.1.2, “Unit Location Label,” on page 329.
V	Virtual Planar. See Section 12.3.1.11, “Virtual Planar Location Label,” on page 331.
W	Worldwide unique ID (for example: Fibre Channel). See Section 12.3.1.9, “Worldwide Unique Identifier,” on page 330.
X	EIA value for empty rack location. See Section 12.3.1.14, “EIA Location Label,” on page 331.
Y	Firmware FRU. See Section 12.3.1.12, “Firmware Location Label,” on page 331.

12.3.1.2 Unit Location Label

The unit location label consists of the prefix “U” followed by uppercase alphabetic characters, digits, and periods (“.”). There is one and only one unit location label in a location code and it is the first element in the location code.

12.3.1.3 Planar Location Label

The planar location label consists of the prefix “P” followed by a non-zero decimal number with no leading zeroes. Planar location labels are present in location codes for planars and all locations on a planar. There is at most one planar location label in a location code. When present, the planar location label immediately follows the unit location label in the location code.

Planars are uniquely labeled within a unit.

12.3.1.4 Air Handler Location Label

An air handler location label consists of the prefix “A” followed by a non-zero decimal number with no leading zeroes. A location code may have zero or more air handler location labels. When present, the air handler location label follows the location label of the resource onto which the air handler is mounted, usually a unit or planar.

Examples of air handlers include blowers and fans.

12.3.1.5 Card Connector Location Label

The card connector location label consists of the prefix “C” followed by a non-zero decimal number with no leading zeroes. A location code may contain zero or more card connector location labels. When present, the card connector location label follows the location label of the resource onto which the card is mounted, usually a planar or another card.

Examples of cards include: Plug-in I/O cards, processor cards, daughter cards, DIMMs, regulator cards, MCMs, L3 cache modules, jumper cards, pass-through interposers, and pluggable module chips.

12.3.1.6 Device Location Label

A device location label consists of the prefix “D” followed by a non-zero decimal number with no leading zeroes. A location code may have zero or more device location labels. When present, the device location label follows the location label of the resource onto which the device is mounted, usually a planar.

Device location labels are used for devices for which a physical location can be determined. These are usually mounted in enclosures that have rigid placement rules and, often, additional hardware support for location determination, such as SES (Storage Enclosure Services) devices.

12.3.1.7 Electrical Location Label

An electrical location label consists of the prefix “E” followed by a non-zero decimal number with no leading zeroes. A location code may have zero or more electrical location labels. When present, the electrical location label follows the location label of the resource onto which the electrical resource is mounted, usually a unit or planar.

Examples of electrical resources include: batteries, power supplies, and chargers.

12.3.1.8 Port Location Label

A port location label consists of the prefix “T” followed by a non-zero decimal number with no leading zeroes. A location code may have zero or more port location labels. When present, the port location label follows the location label of the resource onto which the port is mounted, usually a card or planar.

Examples of resources with a port location label include: ports, connectors, cable connectors, jacks, and interposers.

12.3.1.9 Worldwide Unique Identifier

A worldwide unique identifier location label consists of the prefix “W” followed by a maximum of 16 uppercase hexadecimal digits with no leading zeroes. A location code may have zero or one worldwide unique identifier location labels. When present, the worldwide unique identifier location label follows the location label of the resource that interfaces with the resource having the worldwide unique identifier, usually a port.

12.3.1.10 Logical Path Label

A logical path location label consists of the prefix “L” followed by a decimal or hexadecimal number with no leading zeros. Refer to Section 12.3.2.17, “Port Location Codes,” on page 336 through Section 12.3.2.20, “IDE/ATAPI Device Logical Path Location Codes,” on page 337 to determine when decimal and hexadecimal values are allowed. A location code may have zero or more logical path location labels. When present, the logical path location label follows the location label of the resource that interfaces with the resource being located, usually a port or worldwide unique identifier.

The numeric value portion of the logical path label is a portion of the address, appropriate to the protocol in use, which identifies the resource to be located.

12.3.1.11 Virtual Planar Location Label

A virtual planar label consists of the prefix “V” followed by a non-zero decimal number with no leading zeroes. A location code may have zero or one virtual planar location labels. When present, the virtual planar location label will immediately follow the unit location label in a location code. There is no physical label in the system or any I/O drawer corresponding to the virtual planar location label.

Virtual planars are uniquely labeled within a system.

12.3.1.12 Firmware Location Label

A firmware location label consists of the prefix “Y” followed by a non-zero decimal number with no leading zeroes. A location code may have zero or one firmware location labels. A firmware location code will always contain a unit location label as its first element and a firmware location label its last element.

12.3.1.13 Horizontal Placement Location Label

The horizontal placement location label of an empty space in a rack consists of the prefix “N” followed by a non-zero decimal number with no leading zeroes. A location code may have zero or one horizontal placement location labels. When present, the horizontal placement location label immediately follows the EIA location label.

12.3.1.14 EIA Location Label

The EIA location label of an empty space in a rack consists of the prefix “X” followed by a non-zero decimal number with no leading zeroes. A location code may have zero or one EIA location labels. When present, the EIA location label immediately follows the frame location label.

12.3.1.15 Frame Location Label

The frame location label of an empty space in a rack consists of the prefix “F” followed by a non-zero decimal number with no leading zeroes. A location code may have zero or more frame location labels. When present, the frame location label immediately follows the unit location label.

12.3.1.16 Virtual Function Location Label

A virtual function label consists of the prefix “S” followed by a zero-starting decimal number with no leading zeros. A location code may have zero or more virtual function labels. When present, the virtual function label is appended to the location code of the port that the virtual function uses.

12.3.1.17 Mechanical Location Label

The Mechanical location label consists of the prefix “M” followed by a nonzero decimal number with no leading zeroes. A location code my have zero or more Mechanical location labels. When present, the Mechanical location label follows the location label of the resource onto which the mechanical device is mounted.

Examples of resources with a mechanical location label include: plumbing, valves, and latches.

12.3.1.18 Resource Location Label

The Resource location label consists of the prefix “R” followed by a nonzero decimal number with no leading zeroes. A location code my have zero or more Resource location labels. When present, the Resource location label follows the location label of the parent FRU.

Resource location labels are resources, that are not FRUs, but that need a location code label to be properly identified in the error log. An example of a Resource is a module on the System backplane that is not a separate FRU, but which needs to be separately identified in the system error log for a fail in place type of service strategy.

12.3.2 Converged Location Code Rules

This section describes the rules for building a location code. See also Section 12.3.1, “Converged Location Code Labels,” on page 328 the types of location code labels.

12.3.2.1 Usage of Location Codes

Any reference to a FRU must reference the official location code.

Unofficial internal forms or values of location codes which system components may use for internal convenience are not to be presented to customers, service personnel, etc. They are to be kept internal to those components.

12.3.2.2 Persistence of Location Codes

Physical path location codes are permanent. Physical path location codes will not change unless there has been a change of hardware.

Logical path location codes are dependent on configuration information and are therefore not permanent. Whenever reasonable and possible, logical path location codes will persist across power cycles of the system.

12.3.2.3 Forming Location Codes

Location codes are formed by concatenating one or more location labels together. Location labels start at the largest or most general resource (the unit) and proceed to the most specific in order of containment. The location labels are separated by dashes (“-”).

12.3.2.4 Length Restrictions

Location codes are no more than 79 characters in length. The lengths of individual location labels vary.

12.3.2.5 Location Labels Content

The unit location label may contain uppercase letters, digits, and periods. All other location labels contain only upper case letters and digits. This will avoid problems when printing or displaying the location codes on double byte devices.

12.3.2.6 Physical Representation

In so far as it is possible, location labels must match the labels that are present on the hardware.

Generally, there are labels visible on the hardware to identify connectors, slots, etc. These physical labels must adhere to this specification and must be reflected in the corresponding location codes. This will eliminate the need for the user to translate between the location code presented in logs, displays, reports, and instructions and the location label found on the hardware.

12.3.2.7 Multiple Function FRUs

Some FRUs (Field Replaceable Units) contain more than one logical resource or function. The physical location code refers to the physical FRU. So all the logical resources or functions on a FRU have the same physical location code. Connectors on a FRU have different location codes except for the case of multiple connectors for one port.

12.3.2.8 Multiple Connectors for One Port

Some IOAs have multiple connectors (for example, a D-connector and an RJ-45 connector) for one port. Since there is only one port involved, both connectors have the same location code.

12.3.2.9 Location Label Numbering Scope

For sequentially numbered location labels (planar, card, device, air handler, electrical), each FRU has its own numbering space for its child location labels. That numbering space begins with 1 and increments by 1 for each child location label. Number is in decimal. For example, if there were two planars in a unit, each planar having five card connectors, then each planar would show child location labels of C1, C2, C3, C4, and C5.

This means that for each parent, the child location labels begin with a count of one. As a further example, if there were two adapters in adjacent PCI slots each with two port connectors, the ports on the first adapter would have location labels T1 and T2, and the ports on the second adapter would have location labels of T1 and T2.

Note: Model-specific modifications to this numbering rule may be made, when similar models of a product line are housed in the same enclosure/rack, and the equivalent slots and connectors from each model are lined up with each other as seen by the customer. Then the location code numbering of these equivalent slots and connectors may be the same for each model, even though numbers may be skipped or appear to be out of order on specific models.

12.3.2.10 FRU Orientation

Locations are numbered left to right, top to bottom, back to front with respect to the parent FRU, as viewed from the service position. However, a location label does not change based on the orientation of a piece of hardware within its parent hardware, for FRUs that may have more than one way of being oriented.

If a CEC or I/O drawer is used both in standalone and rack mounted configurations, the rack mounted configuration takes precedence in determining location numbering.

12.3.2.11 Unit Location Codes

The unit location code is the unit location label, Un, for the unit. The unit location label is permanent. The unit location labels may not be etched on the containing racks, since it may not be possible to determine them during manufacture.

A system will have a unit location code composed of the machine type, model, and serial number of the system with the components separated by periods ('.'). For example, a system with machine type 9117, model 250, and serial number 10-ABCDE would have a location code of:

U9117.250.10ABCDE

Some I/O Drawers and CECs may be assigned machine type, model, and serial numbers (MTMS) by manufacturing. Other I/O Drawers and CECs may be assigned feature code, count, and serial numbers by manufacturing. The same requirements for uniqueness apply to both identification schemes. The data is formatted in exactly the same way. And, for the purposes of location codes, the data will be used in exactly the same way.

Drawers and CECs which have a machine type, model, and serial number which the system can obtain will have a unit location code composed of the machine type, model, and serial number with the components separated by periods (“.”). For example, a drawer with machine type 5703, model 012, and serial number 10-30490 would have a location code of:

U5703.012.1030490

Drawers and CECs which have a feature code, count, and serial number which the system can obtain will have a unit location code composed of the feature code, count, and serial number with the components separated by periods (‘.’), for readability. For example, a drawer with feature code 0573, count 001, and serial number 10-40320 would have a location code of:

U0573.001.1040320

Additionally, the count portion of the location code unit label may be modified by firmware at boot time to reflect information about the physical location. It may be modified according to whatever scheme appropriate for the unit's configuration -- provided all other location code rules are followed and that the scheme generates the same value every time when no actual or relative physical location change has been made. For example, TUn (“TU” to indicate the Top CEC Unit, n=1,2,3, or 4 to indicate which drawer numbered top to bottom) may be substituted for the actual count value, so an imaginary CEC following this scheme might have a unit label of:

U1234.TU1.5678901

Enclosure feature code/count must never collide with one another or with any machine type/model. Likewise, enclosure machine type/model must never collide with one another or any enclosure feature code/count.

Drawers for which the system cannot obtain a machine type, model, and serial or a feature code, count, and serial number, will have a location code of the form Uttaa. In this form, the tt is an alphabetic string identifying the kind of drawer, for example, SSA. The aa is a string provided by the rules of the OS to identify the instance of the drawer within the particular system. For example: USSABKUP. The lengths of both tt and aa will vary depending on the kind of drawer. The OS must provide mechanisms by which the customer can ensure the uniqueness of these values.

12.3.2.12 Planar Location Codes

Planar location codes are formed by appending the planar location label, Pn, to the location code of the unit that contains the planar. The Pn value is assigned during the engineering design and manufacturing process, is unique within the unit, is physically displayed on the parent resource, and is present in the VPD of the parent resource. This process is the same irrespective of the kind of planar involved.

Note: In some implementations, a planar is not a separate FRU but is considered to be part of the enclosure. For consistency of the user interface, the enclosure in these cases is still considered to have a location code consisting

entirely of a unit location label. In these cases, the planar has a location code consisting of both the unit location label and the planar location label. This is done even though the planar is not a separate FRU. In these cases, service related VPD and errors will be reported with the planar location code.

12.3.2.13 Card Connector Location Codes

Card connector location codes are formed by appending the card connector location label, Cn, to the location code of the resource to which the card is docked. The location label, Cn, assigned in the engineering and manufacturing process, is unique within the parent resource, is physically displayed on the parent resource, and is present in the VPD of the parent resource.

The process is the same irrespective of the kind of card involved.

12.3.2.14 Riser Card Connector Location Codes

Riser card connector location codes are formed by appending the card location label, Cn, of the child card to the location code of the parent card to which it is attached. For example:

U5702.115.1031010-P3-C4-C2

12.3.2.15 Blade Daughter Card Connector Location Codes

The Blade daughter card slot can actually contain multiple partitionable endpoints depending on the type of daughter card attached. The PCI-E bus can be split (bifurcated) between two PHB's that are separately DLPAR-able (just not hot-pluggable) resources. When that occurs, a -L# suffix is required after the -C# of the daughter card slot to distinctly identify between the two PE's. For example, when the daughter card contains two 4x PCIE devices, the location code for the two PE's might be:

U78A0.001.DNWGDG0-P1-C10-L1

U78A0.001.DNWGDG0-P1-C10-L2

The actual daughter card device ports, in this example, will have the -T# suffix (starting from 1) on top of those. For example, a 4-port PCI-E Ethernet daughter card, might have location codes like:

U78A0.001.DNWGDG0-P1-C10-L1-T1

U78A0.001.DNWGDG0-P1-C10-L1-T2

U78A0.001.DNWGDG0-P1-C10-L2-T1

U78A0.001.DNWGDG0-P1-C10-L2-T2

If only a single PCI device is on the daughter card, then the -L# suffix should not be used, as the -C# is sufficient to identify the PE and device.

12.3.2.16 Virtual Card Connector Location Codes

Virtual card connector location codes are formed as though there were a virtual planar with card slots. For example, a virtual IOA would have a location code of form:

U9117.150.1054321-V5-C2

In the virtual card connector location code, the unit location label specifies the system location code, not the CEC enclosure location code.

It is recommended that the card connector location label have a non-zero numeric part, for human factors reasons.

12.3.2.17 Port Location Codes

Port location codes are formed by appending the port location label, Tn, to the location code of the resource on which the port connector is mounted. The location label, Tn, assigned in the engineering and manufacturing process, is unique within the parent resource, is physically displayed on the parent resource, and is present in the VPD of the parent resource.

12.3.2.17.1 Resources without Port VPD

If a resource without slot map VPD has port numbers physically marked on it, the hard coded slot map will reflect the marked numbers and letters with a “T” prefix. Any letters will be folded to uppercase to avoid double byte display concerns. (Note: only letters 'A' - 'F' may be used in a port location label. Other letters on the card will be ignored and numbers will be assigned for uniqueness.) It is recognized that location labels formed in this way may not conform to the location label rules stated in Section 12.3.1.8, “Port Location Label,” on page 330.

If the port labels are not marked on the standalone adapter, the hard coded slot map will specify location labels of Tn, where n is decimal and equal to the port address (i.e. port 0 will map to T1, port 1 will map to T2, etc.) Whenever possible, the hardware design should be such that this will lead to the preferred left to right, top to bottom, front to back ordering. For standalone adapters that can be installed in multiple systems, ports are labeled beginning with the PCI connector and continuing toward the opposite edge of the card and from tailstock forward toward the opposite edge of the card.

If the adapter is imbedded on a FRU which does not have port location labels marked on it and has ports for other functions, then the numbering to the ports in the hard coded slot map must take into account the other ports on the FRU. The hard coded slot map will comply with the left to right, bottom to top, front to back ordering guidelines. In this case, the first port for the imbedded adapter may be other than T0 and may not be contiguous.

12.3.2.17.2 Determining Port Number

If the port number cannot be determined from VPD or VPD plus configuration or addressing information, software or firmware must infer the port number.

- For SCSI IOAs with multiple PCI configuration spaces, each port has its own configuration space.
- For multiport SCSI IOAs with a single PCI configuration space, firmware or software will add n+1 to the base port's distinguishing value to obtain the port number for the nth port. Thus port 0 (usually closest to the PCI connector edge of the card) will have label T1, port 1 will have label T2, etc.

12.3.2.17.3 Physical Device Location Codes

Devices whose parent supports location label VPD[1] (that is, mounted/docked on a backplane that supports location information for docked devices) will have physical location codes. For example,

U5734.001.10ABCDE-P3-D19

Physical device location codes are formed by appending the physical device location label, Dn, to the location code of the resource to which the device is docked. The location label, Dn, assigned in the engineering and manufacturing process, is unique within the parent resource, is physically displayed on the parent resource, and is present in the VPD of the parent resource.

1.SES devices on SCSI backplanes contain VPD that has a slot map. The slot map associates a SCSI LUN with a location label. Some backplanes that do not actually have SES support have a virtual SES that provides the same function. It is assumed that future protocols will support equivalent function.

Notes:

1. AIX will use logical path location codes for SCSI devices even in situations where a SES device is available to provide device location label information.

2. In some cases, a location code may need to be displayed or logged before the information from the back-plane is available to form the physical location code. In these cases, a logical path location code will be formed according to the applicable rules and used temporarily. Once the information needed to form the physical device label is available, the physical device location code will be used.

12.3.2.18 SCSI Device Logical Path Location Codes -- Real and Virtual

SCSI (Small Computer System Interface) devices whose parent does not support location label VPD will have location codes that are composed of the location code of the controlling SCSI port followed by the SCSI Target (0-15) and SCSI LUN (Logical Unit Number). A SCSI logical-path example is:

U7043.150.1076543-P4-T3-L13-L0
(Decimal L values)

For virtual SCSI, a 48-bit ID is currently used (but is not limited from moving to 64-bit) to identify the attached virtualized SCSI device. This 48/64 bit ID is represented with a -L# in hexadecimal. There is no separate LUN#. Examples are:

U7043.150.1076543-P4-T1-L830000000000
(Hexadecimal L value)

or

U7043.150.1076543-P4-T3-W830000000000-L0
(Hexadecimal W and L values)

12.3.2.19 SAS Device Logical Path Location Codes

SAS (Serial SCSI) devices whose parent FRU does not support location label VPD will have location codes that are composed of the location code of the controlling SAS adapter followed by a series of port labels, “-L#”, for each SAS port traversed from the adapter down to the drive followed by the SCSI LUN (Logical Unit Number). The number value of the “-L#” label will be the lowest phy in the outgoing SAS port with # being a decimal value. For example:

U7043.150.1076543-P1-C3-L3-L1-L5-L0

12.3.2.20 IDE/ATAPI Device Logical Path Location Codes

The desired form of the location code is based on the physical-path location codes obtained from VPD. For example:

U5734.001.1076543-P2-D4

For an IDE device that had an older form of VPD (without physical location codes), then its location code would have looked like:

Ux-Px-(Cx)-Tx-L#
(from -L0 to -L3 where:
L0 = primary/master
L1 = primary/slave
L2 = secondary/master
L3 = secondary/slave)

12.3.2.21 Fibre Channel Device Logical Path Location Codes -- Real and Virtual

Fibre channel devices that are not mounted/docked on a backplane that supports location code VPD will have location codes composed of the location of the port on the controlling IOA followed by the worldwide unique port identifier and LUN. The number value (#) of the "-L#" label is a hexadecimal value. An example FC disk attached to a physical adapter is:

U787A.001.1012345-P1-C5-T2-W123456789ABCDEF0-L1A05000000000000

The same disk being accessed through virtual fibre channel would appear like:

U9111.520.1012345-V2-C4-T1-W123456789ABCDEF0-L1A05000000000000

12.3.2.22 Location Codes for SR-IOV Adapter Virtual Functions

Single Root IO Virtualization allows for multiple "virtual functions" to share the same physical port of a PCI adapter. The -S# suffix, appended to the physical location code of the port (-T#), is used to identify the unique virtual function using that port. The number value (#) of the "-S#" label is a zero starting decimal value determined and managed by the software layer that owns the physical functions of the SR-IOV adapter.

For an SR-IOV ethernet adapter, the third virtual function for the first ethernet port would look like:

U7043.150.1076543-P1-C5-T1-S2

12.3.2.23 Group Labels

Group labels appear on parent FRUs to indicate groupings such as port or DIMM pairs. Group labels are not part of the location label or location code. Group labels may be part of slot map VPD and may be processed by software that displays information on the corresponding FRUs.

12.3.2.24 Sandwich FRU Location Label

A Sandwich FRU has a single location label to describe its location just as any single FRU would.

12.3.2.25 Sandwich FRU Child Location Labels

A Sandwich FRU is like any single FRU in that it has one numbering space for numbering its child locations.

12.3.2.26 Location Code Reported by Sensors

The location code reported by a sensor is the location code of the FRU being monitored by the sensor.

12.3.2.27 Sensor Locations

The location code of a sensor is the location code of the FRU on which the sensor is located.

12.3.2.28 Location Code Reported for Indicators

The VPD that reports an indicator will give the location label of the resource identified by the indicator, not the location label of the indicator itself.

12.3.2.29 Indicator Locations

The location code of an indicator is the location code of the FRU on which the indicator is located.

12.3.2.30 Firmware Location Codes

The location specified for firmware is left to the platform except that the location code must match the scope of the firmware and the location code must follow the form specified above, beginning with a unit location label and ending with a firmware location label. For example, firmware for port 2 (T location label value) on planar 1 could have a location code of the form:

U7879.001.1054321-P1-Y2

If the firmware is considered to be system wide, then the planar location label would not be present and the unit location label specifies the system location code not the CEC enclosure location code:

U9117.001.1054321-Y1

12.3.2.31 Bulk Power Assembly (BPA) Location Codes

The unit location label for a BPA and its components consists of the “U” prefix and the MTMS of the rack of which the BPA is a component.

In some configurations, there are two BPAs in one rack. If they were treated separately, the second BPA would have the same location label as the first which would lead to location code collisions. Therefore, from the perspective of location codes, the two BPAs will be treated as on BPA. The planar location labels and, when necessary, other location labels within the second BPA are incremented so that they are not the same as the labels in the first BPA. For example, the front side BPA has a planar P1 and the back side BPA has a planar P2 etc.

12.3.2.32 Internal Battery Features Location Codes

The unit location label of an Internal Battery Feature (IBF) is the unit location label of the BPA to which it belongs. The planar location labels and, if necessary, other location labels within the IBF are incremented so that they are not the same as the labels in the BPA or any other IBF belonging to that BPA.

12.3.2.33 Media Drawer Location Codes

In some configurations, there are media drawers that do not have a MTMS. The unit location label for these media drawers is the unit location label of the CEC to which it belongs. The planar location labels and, if necessary, other location labels within the media are incremented so that they are not the same as the labels in any other media drawer in the system or any labels in the CEC.

12.3.2.34 Horizontal Placement Location Labels

The horizontal placement location label begins with the prefix “N” followed by the digit “1” for the left side of the frame (viewed from the front) or the digit “2” for the right side of the frame (when viewed from the front).

12.3.2.35 EIA Location Label

The EIA location label begins with the prefix “X” followed by a nonzero digit which represents the EIA location of the bottom of the unit that is or would be in the frame at that location.

12.3.2.36 Blade Chassis Location Codes

The disk storage module in a blade chassis may consist of a backplane and a set of disks that plug into the backplane. Such a storage module is assigned an enclosure feature code which is used to define the unit location label (Un) of a disk in a storage module.

In some blade chassis there may be multiple identical storage modules. All storage modules have a unit location with the same enclosure feature code. The backplane within the left storage module has label P1, the backplane in the next storage enclosure has label P2, and so on, to help with disk identification.

12.3.2.37 Location Codes for Hot-pluggable Devices

There are multiple occasions where devices may be added after the system has booted and the addition did not use any dynamic reconfiguration flows. In this situation, the O/S does not have adequate information from the device tree's **"ibm,loc-code"** properties to construct a correct physical location code. One example of this is with IDE/SCSI drives that are hot-pluggable, where the O/S only knows the logical location code from the IDE/SCSI bus. Firmware, however, may know the physical location code of the drive based on the unit-address. Similarly, a USB root-hub may have multiple down-facing ports with different physical location codes, but the root-hub is only a single device tree node, so only a single location code can be provided with just the **"ibm,loc-code"** property.

The **"ibm,loc-code-map"** property contains a list of pairs (unit-address, location code), both as encoded strings. It describes the physical location code for each potential child node.

R1–12.3.2.37–1. The instance of each USB root-hub in the device tree must contain the **"ibm,loc-code-map"** property if the root-hub has multiple down-facing ports. This applies to both the OHCI and EHCI interfaces of a USB adapter. Lack of this property implies there is only a single down-facing root-hub port from that USB interface.

R1–12.3.2.37–2. To determine the location code for an end device associated with leaf open firmware node, the O/S must use the **"ibm,loc-code-map"** entry with a matching unit-address, if it exists, in preference to the parent's **"ibm,loc-code"** property.

R1–12.3.2.37–3. The **"ibm,loc-code-map"** property must not contain an entry for a port from an embedded IOA that is not externally connected, or if the location code is undeterminable.

R1–12.3.2.37–4. If the unit-address architecture for certain node types do not strictly bind a particular unit-address with a hardware location, the **"ibm,loc-code-map"** property must not exist in the parent of those nodes.

12.3.2.38 Location Code for USB Attached Devices

The root hub port number used determines the location code up to the Tn suffix. There can be zero or many intervening hubs, and the intervening "Ln" location labels are in path order. All immediate children are of root hubs are represented by "L1". For devices not directly attached to the root hub, the "Ln" will correspond to the software port number, n, of the parent hub that a device is attached to, counting from 1.

For example, the location code for a USB device attached to an IOA imbedded on a backplane with location label P1, using port T5, and with an intervening hub under which the device is attached to the third port would be:

U787A.001.10ABCDE-P1-T5-L1-L3

12.4 Vital Product Data

12.4.1 Introduction

The set of all Vital Product Data (VPD) from the FRUs of a system is the product topology information which uniquely defines that system's hardware and firmware elements. The system VPD describes a system in terms of various FRUs, part numbers, serial numbers and other characterizing features. With VPD, mechanisms may be provided for collecting information such as field performance and failure data on any FRU in a system. Also, with the feedback from the field into an installed system data base, the delivery of complete and accurate Miscellaneous Equipment Specifications (MESs) to customers can be assured.

R1–12.4.1–1. FRUs used in LoPAPR platforms must provide machine-readable VPD as defined in Section 12.4, "Vital Product Data," on page 341.

R1–12.4.1–2. LoPAPR platforms must support the collection of, and provide availability to, Vital Product Data.

Platform Implementation Notes:

1. It is the intent of this architecture that the FRUs of a system be self describing using VPD.
2. There are FRU's which have VPD which is not in the format described herein, such as JEDEC. In the case of use of these parts, the firmware may choose to translate the FRU VPD data into the LoPAPR format when the OF device tree is being generated. For I/O adapters which have different VPD, their device driver must perform the reading and translation.

12.4.2 VPD Data Structure Description

Architecture Note: Even though only a few large and small resource tags have been defined (see the *PCI Local Bus Specification* [18]), the current definition will allow all possible tags except for (0x00). This will allow later devices with a new, previously undefined tag, to work.

R1–12.4.2–1. Vital Product Data when reported to the OS via *ibm,get-vpd* or the OF device tree, must conform to the data structures describe in the *PCI Local Bus Specification* [18], section 6.4[1], except as follows:

- The VPD will consist of only the following sequence of tags: Large Resource type identifier string tag (type 2, byte 0 = 0x82) with FRU name, Large Resource type VPD keywords tag (type 16, byte 0 = 0x90) with FRU VPD keywords, Small Resource type end tag (type 15) with or without checksum covering the above.
- Only resource tags, lengths and checksums are binary. All other data must be in ASCII format.

Architecture Note: There are three keywords (FU, SI and VI) which allow binary data. There are no other exceptions. Also, the length code following a large resource tag is Little-Endian. That is, the first byte is the low order byte and the second byte is the high order byte of the length.

Implementation Notes:

1. Version 2.2 of the *PCI Local Bus Specification* changed the format and location of VPD information. With that change, a device with Version 2.2 VPD will result in no VPD being detected by the firmware. In this case the selected device driver will have to access and reformat the VPD information so that the format of the data provided to the OS is in the format which is required by the OS.

1. The PCI 2.1 VPD keywords are PN, FN, EC, MN, SN, LI, RL, RM, NA, DD, DG, LL, VI, FU, SI, and Z0-ZZ

2. The definition of a small resource tag is where bit 7 is 0. Then bit 6 through 3 are the type and bits 2 through 0 are the length. An end tag is a type of 15. Thus a 0x78 is a small resource end tag of length 0 and 0x79 is a small resource end tag of length 1. The valid end tags are 0x78 through 0x7F.

R1–12.4.2–2. The end tag checksum, when provided to the OS, must cover all resources beginning with the first byte (a resource tag) up to, but not including, the Small Resource Type End tag.

R1–12.4.2–3. If the platform determines that the VPD that it has collected is invalid, then the platform must discard any collected data and replace it with:

- Large Resource type identifier string tag (type 2, byte 0 = 0x82) with value 'NOT_VALID_VPD'.
- Large Resource type VPD keywords tag (type 16, byte 0 = 0x90) containing the "YL" keyword and associated location code data.
- Small Resource Type End tag with or without a checksum covering the above.

R1–12.4.2–4. If the device VPD is valid, all of the device VPD must be transferred including the Small Resource Type End Tag.

R1–12.4.2–5. A "YL" keyword must be added to the Large Resource type VPD keywords tag (type 16, byte 0 = 0x90) when the VPD is reported to the OS via either the *ibm,get-vpd* RTAS call or the OF device tree.

The checksum byte after the (0x79) resource tag will cause the binary sum of all the bytes from the first large resource tag, carries being discarded, to result in 0x00. As noted in Requirement R1–12.4.2–2, the (0x79) small resource tag is not included in this sum.

12.4.3 Keyword Format Definition

The exact format of the VPD is vendor-specific but falls within the specification as defined in the following subsections.

12.4.3.1 VPD fields

The fields defined in Table 160, "LoPAPR VPD Fields," on page 343 are stored in VPD modules at the time of manufacturing.

R1–12.4.3.1–1. LoPAPR platforms and FRUs must provide VPD fields marked as required in Table 160, "LoPAPR VPD Fields," on page 343 and for all VPD fields provided must adhere to the definitions specified by Table 160.

R1–12.4.3.1–2. Each system must have VPD that contains the system's SE keyword (see Requirements R1–12.2–1, R1–12.2–3, R1–12.2–12, and R1–12.2–14) and the system's TM keyword (see Requirements R1–12.2–1, R1–12.2–3, R1–12.2–13, and R1–12.2–14). The description (large resource type of 0x82) for this VPD in the **`"ibm,vpd"`** property must be "System VPD".

R1–12.4.3.1–3. VPD for memory FRUs must contain the SZ keyword.

Platform Implementation Note: There are circumstances where vendor preference or manufacturing processes may require that some fields be omitted (for example; Serial Number). This should be treated as an exception and

should be accompanied by an appropriate level of risk assessment. In the case of an SN exception, the SN should be omitted, not made blank, NULL, or a fixed value.

Table 160. LoPAPR VPD Fields

Keyword	Data Type	Data Length (Bytes)	Required?[a]	Description
A1	ASCII	Up to 16	--	Storage Facility Image MTMS - Logical ID Length (MTMS-a####). Linkage between logical entities. Examples: Used on Disk FRU resource to link disk to array site. Used on array site resource to link array site to array. Used on array resource to link array to rank. Used on rank resource to link rank to segment pool.
AC	ASCII	Up to 32	--	Account Name Used on HMC resource.
AD	ASCII	Up to 10	--	Addressing Field
AP	ASCII	Up to 16	--	Asset Protection Key Password
AS	ASCII	Up to 8	--	Reserved
AT	-	-	--	Reserved
AX	ASCII	Up to 32	--	AIX name
B1	ASCII	A multiple of 16, up to a max of 240	--	Contains 1 to 15 instances of the following 16 byte definition, concatenated together. Contains the base ethernet MAC address and an instance count. The count specifies the number of valid MAC addresses starting with the base MAC address and incrementing by one for each successive MAC address, until the specified number of MAC addresses have been created. The field contains the ASCII coded hexadecimal representation of the binary value, as follows: Bytes 1 - 12 Base MAC address 13 - 14 Reserved (ASCII “00”) 15 - 16 Count
B2	ASCII	A multiple of 32, up to a max of 224	--	Contains 1 to 7 instances of the following 32 byte definition, concatenated together. SAS WWIDs. The Count field is the number of available WWIDs starting from the base and incrementing by 1 for each new WWID. The field contains the ASCII coded hexadecimal representation of the binary value, as follows: Bytes 1 - 16 Base SAS WWID 17 - 24 Reserved (ASCII “00000000”) 25 - 32 Count
BC	ASCII	Up to 12	--	Bar Code
BH	ASCII	2	--	BatcH code - used for vintage if no serial number

Table 160. LoPAPR VPD Fields ***(Continued)***

Keyword	Data Type	Data Length (Bytes)	Required?[a]	Description
BR	ASCII	2	--	Brand keyword value “xy” **Note:** The following are the values currently defined. To get other product specific values, go through the normal LoPAPR change process. x = Type of machine ◆ “B” - Reserved ◆ “C” - Reserved ◆ “D” - Reserved ◆ “I” - Reserved ◆ “N” - Reserved ◆ “O” - Reserved ◆ “P” - Reserved ◆ “S” - IBM Power Systems™ platforms ◆ “T” - OEM Power Systems platforms ◆ Other values are reserved. y = Specific Information ◆ “0” - no specific information ◆ Other values are reserved.
BT	ASCII	10	--	Battery Replacement date in YYYY-MM-DD format. Used on Primary Power Supply FRU in rack enclosure.
CC	ASCII	4	Required when a CCIN is required by code to configure or service the component	Customer Card Identification Number (CCIN)
CE	ASCII	1	--	CCIN Extender
CD	ASCII	Up to 10	--	Card ID
CI	ASCII	16	--	CEC ID - of the CEC, that is the logical controller of an MTMS (machine-type-model-serial #), like a drawer. The 16 byte CI field definition is TTTT-MMM SSSSSSS TTTT-MMM - is an 8 byte field. The high order 4 bytes are the system type, followed by a dash, followed by the 3 low order bytes which is the model. blank - is a 1 byte separator character, that separates the type-model from the serial #. SSSSSSS - is the 7 byte serial number. The high order 2 bytes are the “plant of manufacture” and the low order 5 bytes are the “sequence number”.

Table 160. LoPAPR VPD Fields ***(Continued)***

Keyword	Data Type	Data Length (Bytes)	Required?[a]	Description
CL	ASCII	Up to 32	--	Code Level, LID keyword Format: fix pack MIF name, “space”, Load ID (8 characters), time stamp (12 characters) The fix pack name must be the same as the name used in the MI keyword and is delimited by a space. The time stamp is, hours (2 characters) + Minutes (2 characters) + Year (4 characters) + Month (2 characters) + Day (2 characters). The LID level reported is the current active level. **Note:** There is no correlation between the CL keyword value and which of the 3 candidate fixpack levels are reported in the MI keyword.
CN	ASCII	Up to 7	--	Customer Number
CV	ASCII	Up to 4	--	Country Number
DC	ASCII	2 1 12	--	Action Code, timestamp blank (space) **TimeStamp**: yyyymmddhhmm Action Codes: BD = Build Date AM = added as MES AB = added as bulk MES AI = available at install ID = field install date AC = added with field EC AU = added from unknown source AR = added in repair action AT = added temporarily in field AH = added manually in field Returned on history file: RU = removed unknown (field) RR = removed in repair action RC = removed with EC RT = removed temporarily / powered off (field) RM = removed permanently (field) RN = removed to another system
DD	-	-	--	Reserved -- used for MicroChannel Architecture VPD
DG	-	-	--	Reserved -- used for MicroChannel Architecture VPD
DU	ASCII	Up to 10	--	Drawer Unit
DL	ASCII	Up to 10	--	Drawer Level
DP	ASCII	Up to 255	--	Disk Space Characteristics Used on Disk resource. Comma separated string. Example (in the following, the “<“, “>”, and text in between these would be replaced by a value representing the quantity specified by the words between “<” and “>”): “VN=<Volume Serial Number>, VD=<Vendor>, DT=<Disk Type>, DC=<Disk-Capacity>, DR=<Disk-RPM>, DS=<Disk-Interface-Speed>, AP=<Array-Site-Position>, ST=<Status>”
DS	ASCII	Up to 30	--	Displayable Message (if not defined, created by the contents of the ID large resource (82))
EA	ASCII	Up to 24	--	Electronic Message for electronic customer support (ECS)

Table 160. LoPAPR VPD Fields ***(Continued)***

Keyword	Data Type	Data Length (Bytes)	Required?[a]	Description
EC	ASCII	12	Required if the part number is not changed with every modification	Engineering Change Level (technical features, revision level, vintage level)
ET	ASCII	2	--	Enclosure Type. This value is defined in the *System Management BIOS (SMBIOS) Reference Specification* [24]. The ASCII value specified below is the ASCII representation of the hexadecimal representation of the byte value defined in the SMBIOS specification (for example, 0x0B becomes “0B”). Bit 7 of the byte number is the chassis lock bit (if present value is a 1, otherwise if either a lock is not present or it is unknown if the enclosure has a lock, the value is a 0). If the chassis lock bit is set, the following get changed to the corresponding ASCII representation (for example, “01” becomes “81”, “10” becomes “90”) “01” Other “02” Unknown “03” Desktop “04” Low Profile Desktop “05” Pizza Box “06” Mini Tower “07” Tower “08” Portable “09” LapTop “0A” Notebook “0B” Hand Held “0C” Docking Station “0D” All in One “0E” Sub Notebook “0F” Space-saving “10” Lunch Box “11” Main Server Chassis “12” Expansion Chassis “13” SubChassis “14” Bus Expansion Chassis “15” Peripheral Chassis “16” RAID Chassis “17” Rack Mount Chassis “18” Sealed-case PC
FC	ASCII	8	--	The Feature Code is 8 bytes formatted as 4 characters, a dash, and three characters.
FD	ASCII	7	--	Field Bill of Material (FBM) EC level
FG	ASCII	4	--	FlaG Field: The first two bytes contain a VPD flag in the form of VS. V=V indicates that there is VPD. V=N indicates there is no VPD. S=S indicates that the VPD contains a slot map. S=P indicates there is a port map. S=N indicates no slot map or port map. S=B indicates both a slot map and a port map. The right two characters contain the FRU identification keyword.
FI	ASCII	2 - 8	--	Frame ID: 2 hex byte value for SPCN or 8 character logical frame number for DASD backplane.
FL	ASCII	Up to 16	--	FRU Label: This is a variable length ASCII character data area for the FRU Label.
FN	ASCII	Up to 8	Required	FRU Number (Board, card, or assembly Field Replacement Unit number).
FU	Binary	Up to 10	--	Function Unit - This function identifies which function in a multi-function IOA this VPD data applies to. Only one FU field can appear per VPD tag. Data is binary encoded.

Table 160. LoPAPR VPD Fields ***(Continued)***

Keyword	Data Type	Data Length (Bytes)	Required?[a]	Description
H1	ASCII	1	--	Partition HSL Pool Used on a partition resource that is part of a storage facility image.
ID	ASCII	2	--	Two ASCII characters are used to identify each system in a Storage Facility. Valid values are 00 and 01.
IF	ASCII	Up to 16	--	Storage Facility MTMS - InterfaceID Length (MTMS-####). Identifies one or more adapter I/O interfaces in the storage facility that interconnect device adapters and storage enclosures. Used on Device Adapter FRUs in I/O enclosure and on Storage Enclosures. Device Adapters have two Interfaces Ids (comma separated string), and storage enclosures have one. The number could change for future products.
L1 L2 L3 L4 L5 L6	ASCII	Up to 30 Up to 30 Up to 30 Up to 10 Up to 30 Up to 12	--	Location Information: Individual or Company Name Street Address City, State, Country Zip Code Contact Name Contact Phone Number
LA	ASCII	32	--	LIC Node Alternate Bus VPD: This fixed format data field contains 32 bytes of LIC I/O node alternate bus VPD data.
LE	ASCII	Up to 17	--	LIC VMRF Example: SEA 5.1.0.0345. Used on a partition resource that is part of a storage facility image or on an enclosure or enclosure resource that has a firmware level.
LI	ASCII	Up to 10	--	Adapter Software Identification
LL	ASCII	Up to 10	--	Adapter Software Level
LN	ASCII	32	--	LIC Node VPD: Fixed format 32 bytes of VPD data.
LO	ASCII	2	--	Location (INternal/EXternal)
LP	ASCII	32	--	LIC Node Primary Bus VPD: Fixed format 32 bytes of VPD data.
LS	ASCII	Up to 255	--	Logical Space Characteristics Used on Storage Logical resources. Examples: Used on Array resource. Comma separated string. Including: “AN=Array Serial Number, AT=Array-Type, AC=Array Configuration, Rank Position”. Used on Rank resource. Comma separated string. Including: “RN=Rank Serial Number, ST=Segment-Type(FB-1G, CKD-Mod1), SS=Segments(####), SU=Segments-Used, RG=Rank Group(#)”. Used on Segment Pool resource. Comma separated string. Including: “ST=Segment-Type(FB-1G, CKD-Mod1), SS=Segments(####), SU=Segments-Used, VS=Virtual-Segments(####), VU=Virtual-Segments-Used(####)”, RG=Rank-Group(#).
MD	ASCII	4	--	Model Number: 3 characters with a leading blank.
MF	ASCII	2	--	Map Format: Two hex characters identify the slot or port map format that follows. This keyword must immediately precede the SM or PM keyword.

Table 160. LoPAPR VPD Fields ***(Continued)***

Keyword	Data Type	Data Length (Bytes)	Required?[a]	Description
MI	ASCII	Up to 40	--	Microcode Image This keyword is only used to describe dual sided alterable system firmware Image which can be altered by the OS via the *ibm,update-flash-64-and-reboot* RTAS call. Both the *ibm,validate-flash-image* and the *ibm,manage-flash-image* RTAS calls are supported. May or may not be able to be updated via non-OS visible means. Format: t side Microcode Image name, “space”, p side Microcode Image name, “space”, booted Microcode Image name The Microcode Image name must be a 9 character name of the form: **“NNSSS_FFF”** where “NN” is a two character name (assigned by the GFW Firmware Distribution Coordinator) used to identify a set of platforms; “SSS” is a 3 character code stream indicator; “_” is a separation character for readability; and, “FFF” is a 3 character identifier of the current microcode level. **Note:** The booted fix pack level reported may not match the current p-side or t-side level as a result of concurrent update. The important information is what is in flash, and that is what is being reported. The hypervisor will have to cache the value for the booted level, and should go to FSP to check what's current on flash for the p-side and t-side levels.
ML	ASCII	Up to 40	--	Microcode Level This keyword is only used to describe dual sided alterable system firmware images which can be altered by the OS via the ibm,update-flash-64-and-reboot RTAS call. Both the ibm,validate-flash-image and the ibm,manage-flash-image RTAS calls are supported. May or may not be able to be updated via non-OS visible means. Format: t side Microcode Level name, “space”, p side Microcode Level name, “space”, booted Microcode Level name The Microcode Image name must be a 8 character name of the form: **“FWVRE.MF”** where "FW" is a static prefix representing 'firmware' “V” is Version - Power Processor Level "R" is Revision - Typically GA number from the processor level "E" is Extension - Typically zero. If non-zero, designates off cycle single system release "M" is Modification - associated Service Pack Level (0-Z) "F" is Fix Level - Typically zero. If non-zero, designate off cycle, targeted fixes **Note:** The booted fix pack level reported may not match the current p-side or t-side level as a result of concurrent update. The important information is what is in flash, and that is what is being reported. The hypervisor will have to cache the value for the booted level, and should go to FSP to check what's current on flash for the p-side and t-side levels.
MN	ASCII	10	--	Manufacturer ID (source of device, name and location)
MP	ASCII	3	--	Module Plug count. This counter keeps track of the numbers of times a module has been plugged, so that reliability statistics can be kept.
MS	ASCII	Up to 6	--	MES Number

Table 160. LoPAPR VPD Fields ***(Continued)***

Keyword	Data Type	Data Length (Bytes)	Required?[a]	Description
MU	ASCII	32	--	Machine Universal Unit ID (UUID). The value is the ASCII coded hexadecimal representation of the 16 byte binary value.
N5	ASCII	Up to 228	--	Processor CoD Capacity Card Info
N6	ASCII	Up to 231	--	Memory CoD Capacity Card Info
N7	ASCII	Up to 144	--	Processor on Demand billing data
N8	ASCII	Up to 145	--	Memory on Demand billing data
NA	ASCII	Up to 16	--	Network Address (ASCII coded hexadecimal representation of the binary value.)
NC	ASCII	Up to 25	--	This keyword is used to describe the prefix name of the file used to install a single sided alterable system firmware or adapter/device microcode image. When this keyword is present, the complete file name will be described by the content of the NC keyword, concatenated with ".", concatenated with the content of the RM keyword.
NN	ASCII	16	--	World Wide Node Name - IEEE assigned 64 bit identifier (16 hexadecimal digits) for Storage Facility. Valid values are 0-9, A-F.
NT	ASCII	Up to 32	--	Sub-machine type
NV	ASCII	Up to 24	--	NVRAM ID, part number, location and size
NX	-	-	--	Reserved
OS	ASCII	Up to 17	--	OS name and level. The OS level is shown in the form of: V.R.M.F where V is the version, R is the release, M is the modification, and F is the fix.
PA	ASCII	1	--	Op Panel installed flag. "Y" = yes a panel is installed, "N" = no a panel is not installed. The absence of this keyword means, that an Op Panel is installed.
PC	ASCII	Up to 16	--	Processor Component Definition
PD	ASCII	Up to 8	--	Power dissipation/consumption
PI	ASCII	Up to 8	--	Processor ID or unique ID (used for licensing control)
PL	ASCII	Up to 32	--	Location code
PM	ASCII	Up to 16	--	Port Map: Contains the RIO Port Map label information. Must be preceded by the MF keyword.
PN	ASCII	12	Required if it is different from the FRU number (FN)	Part number of assembly.
PO	ASCII	Up to 16	--	SPCN VPD: Identification of the SPCN VPD area on the I/O backplane.
PP	ASCII	32	--	Power Parameters: SPCN field identifying Power node parameters.
PR	ASCII	16	--	Power: 16 ASCII hexadecimal characters that represent the 8 bytes of binary information for Power Control.
R1	ASCII	Up to 16	--	Rack MTMS - Rack Location Length (MTMS-Exx). - used on storage enclosure resources.

Table 160. LoPAPR VPD Fields ***(Continued)***

Keyword	Data Type	Data Length (Bytes)	Required?[a]	Description
R2	ASCII	1	--	Rack Number Used on Rack enclosure resources. Provides an ordered number of racks in the storage facility.
RA	-	-	--	Reserved
RD	ASCII	16	--	Power Domain ID - is the MTMS of the BPA, that powers an MTMS (Machine-Type-Model-Serial #), like a CEC or drawer. The 16 byte RD field definition is: TTTT-MMM SSSSSSS TTTT-MMM - is an 8 byte field. The high order 4 bytes are the system type, followed by a dash, followed by the 3 low order bytes which is the model. blank - is a 1 byte separator character, that separates the type-model from the serial #. SSSSSSS - is the 7 byte serial number. The high order 2 bytes are the "plant of manufacture" and the low order 5 bytes are the "sequence number".
RI	ASCII	4	--	Power Resource ID: A 4 byte hex field providing a unique logical ID for the power resource.
RJ	ASCII	Up to 16	--	RIO-G Loop Used on I/O enclosure resource. Identifies RIO-G loop on the reporting system.
RK	ASCII	16	--	Rack Unique ID - is the 64 bit Dallas "1-wire" unique ROM code. The first 8 bits are a 1-Wire family code. The next 48 bits are a unique serial number. The last 8 bits are a CRC of the first 56 bits. Each of the 16 4-bit nibbles are converted to 16 ASCII characters, in the range 0-9 or A-F.
RL	ASCII	Up to 24	--	This keyword is only used to describe single sided non-alterable system firmware or adapter/device microcode image which can not be altered by any means; OS visible or non-OS visible. A single alphanumeric character string that defines the level of the image. ROM id, Location ID, ROM part number, FW part number, FW level, FW code, release date, FW size.
RM	ASCII	Up to 25	--	This keyword is only used to describe single sided alterable system firmware or adapter/device microcode image which can be altered by the OS. The OS alters the system firmware image via the *ibm,update-flash-64-and-reboot* RTAS call. Neither the *ibm,validate-flash-image* nor the *ibm,manage-flash-image* RTAS calls are supported. May or may not be able to be updated via non-OS visible means. A single alphanumeric character string that defines the level of the image. ROM id, Location ID, ROM part number, FW part number, FW level, FW code, release date, FW size
RN	ASCII	Up to 2	--	Rack Name
RP	ASCII	1	--	RIO-G Position Offset Used on I/O enclosure resource. Identifies the distance in enclosures from the reporting system - first enclosure is offset 1.
RS	ASCII	Up to 128	--	IBM LoPAPR Compliant platform unique VPD: Start of a data area.
RT	ASCII	4	--	Record Type. Contains a four character Record Name that represents a VPD Definition. The following list associates Record Names with their VPD Definitions. ◆ "VSYS" - System MTMS VPD ◆ "VCEN" - Enclosure MTMS VPD ◆ "VINI" - FRU VPD record
RW	-	-	--	Reserved
RX	ASCII	Up to 25	--	This keyword is only used to describe single sided microcode image which can not be altered by the OS, but may be updated via non-OS visible means. A single alphanumeric character string that defines the level of the image.
S1	ASCII	Up to 8	--	Serial Number of attached machine

Table 160. LoPAPR VPD Fields ***(Continued)***

Keyword	Data Type	Data Length (Bytes)	Required?[a]	Description
S3	ASCII	16	--	Storage Facility MTMS Length (MTMS). Used on system resource, partition resources, array-site, resources, array resources, rank resources, storage pool resources, and enclosure resources in a storage facility to identify the storage facility.
S4	ASCII	Up to 16	--	Storage Facility Image MTMS Length (MTMS). Used on partition, array-site, array, rank, and storage pool, and enclosure-FRU resources, associated with a storage facility image.
SC	ASCII	Up to 44	--	Specify codes
SE	ASCII	Up to 7	See Requirements R1–12.2–1, R1–12.2–3, R1–12.2–12, R1–12.2–14, and R1–12.4.3.1–2	Machine or Cabinet Serial Number.
SF	ASCII	Up to 8	--	Field Change Shipping Instruction (FCSI) number
SG	ASCII	7	--	2 high order bytes are the “plant of manufacture code”, followed by the low order 5 bytes, which are the “unit sequence number”. This unit sequence number must be DDDD0, ADDD0, AADD0, AAAD0, or AAAA0 where D is a digit 0-9 and A is an alphabetic character A-Z excluding E, I, J, O, Q, S, U. The right most character of the unit sequence number must be set to 0.
SI	Binary	Up to 10	--	Subsystem Vendor ID. Data is binary encoded.
SL	-	-	--	Reserved
SM	ASCII	Up to 16	--	Slot Map: Contains Slot Map information. Must be preceded by the MF keyword.
SN	ASCII	12	Required and must be unique for each part with the same FN	Serial Number 12 characters.
SU	ASCII	12	See notes 1	The System Unique Identifier (SUID) is a twelve hexadecimal character value that is unique to a given system anywhere in the world. The number range is obtained from the Institute of Electrical and Electronic Engineers. There are no IBM asset protection considerations involved.
SY	ASCII	7	--	System Number (only one allowed)
SZ	ASCII	Up to 10	For memory FRUs, see Requirement R1–12.4.3.1–3	Size in decimal, with no leading zeroes. For memory (for example, cards and DIMMs), it provides the memory size in MB’s. As an example, a 32 GB memory card would be coded as 32768.
Tl	ASCII	8	--	Attached machine type-model
TM	ASCII	8	See Requirements R1–12.2–1, R1–12.2–3, R1–12.2–13, R1–12.2–14, and R1–12.4.3.1–2	The high order 4 bytes are the “Machine Type”, the next byte is a dash, followed by the 3 byte “Model”.
TN	ASCII	8	--	The high order 4 bytes are the “Machine Type”, the next byte is a dash, followed by the 3 byte “Model”.

Table 160. LoPAPR VPD Fields ***(Continued)***

Keyword	Data Type	Data Length (Bytes)	Required?[a]	Description
U1	ASCII	Up to 255	--	Logical Configuration String Used on a partition resource that is: CKD3380=####, CKD3390=####, SCSI512=####, SCSI520=####, SCSI Host Ports=####, SCSI LUN Groups=####. Used on device adapter or host adapter FRU for logical configuration information, if any. Used on Storage Enclosure. Comma separated string including "BA=Base Address(xx)".
U2	ASCII	Up to 255	--	Host Inventory Used on a partition resource that is part of a storage facility image for storage facility image logical configuration. Comma separated string. Including: "Host OS Type1=####, Host OS Type 2=####, . . .".
U3	ASCII	Up to 255	--	Reserved for future use. Used on partition resource or Storage Configuration resources.
U4	ASCII	Up to 255	--	Reserved for future use. Used on partition resource or Storage Configuration resources.
U5	ASCII	Up to 255	--	Reserved for future use. Used on partition resource or Storage Configuration resources.
U6	ASCII	Up to 255	--	Reserved for future use. Used on partition resource or Storage Configuration resources.
U7	ASCII	Up to 255	--	Reserved for future use. Used on partition resource or Storage Configuration resources.
U8	ASCII	Up to 255	--	Reserved for future use. Used on partition resource or Storage Configuration resources.
U9	ASCII	Up to 255	--	Reserved for future use. Used on partition resource or Storage Configuration resources.
VE	-	-	--	Reserved -- used for MicroChannel Architecture VPD
VI	Binary	Up to 10	--	Vendor ID / Device ID Only one VI may appear per VPD tag.
WN	ASCII	16	--	Contains the ASCII coded hexadecimal World Wide Port Name (WWPN).
YK	ASCII	4	See Requirement R1–12.2–3	Ties together multiple enclosures that share the same combined SE and TM.

a. A lack of a hard requirement in this column does not mean that the keyword is never required; only that it is not required all the time. Keywords which are not marked as "Required" are required when appropriate for the specific keyword.

Notes referenced in Table 160, "LoPAPR VPD Fields," on page 343:

1. When the BR keyword indicates "S" or "T" for the type of machine, this field must be present and must be non-blank.

Implementation Note: The following keywords are defined in this architecture with the same or similar usage: AD, CD, DC, DL, DS, DU, EC, FC, FN, LO, NA, P C, PI, PN, RL, RN, SN, SZ, TM, and US.

12.4.3.2 Additional Fields for Product Specific use

Three fields are available for user, system or product specific data for which no unique keyword has been defined. The first and second ranges of fields in the list are not addressed in the PCI Specifications and are unique to LoPAPR platforms.

1. U0 - UZ User specific
2. N0 - NF Reserved
   V0 - VF Reserved

3. Y0 - YZ System Information specific

4. Z0 - ZZ Product specific

Note: If firmware/software has a specific need for a keyword, then it must be provided by the appropriate component VPD.

The Y? and Z? fields defined in Table 161, “LoPAPR Usage of Product Specific VPD Fields,” on page 353 are specific to LoPAPR platforms. As a need for these fields is determined, they will be defined in this document. The table contains several keywords which have been defined as place holders.

Table 161. LoPAPR Usage of Product Specific VPD Fields

Keyword	Data Type	Data Length (Bytes)	Description
N5	ASCII	Up to 56	Processor CoD Capacity Card Info per Section 7.3.16.4.1, “CoD Capacity Card Info,” on page 213
N6	ASCII	Up to 56	Memory CoD Capacity Card Info per Section 7.3.16.4.1, “CoD Capacity Card Info,” on page 213
U?	ASCII	Up to 128	User Data:? = 0...9, A...Z
Y0	ASCII	64 2 2 24 24 12	Board Repair Actions: # times board repaired # times board updated with code patches Copy of system ID Y2 field Copy of system ID TM field Copy of system ID PI field
Y1	ASCII	24 2 2 8 12	Error Descriptor: # allowable messages # valid messages Messages of 20 bytes each, first, last and most recent 4 messages Error Code Date/Time Stamp: yyyymmddhhmm
Y2	ASCII	24 4 4 4 12	Only in system VPD EEPROM: Manufacturer’s Location Machine Type Model ID Cabinet Serial Number
YK	ASCII	Up to 4	Ties together multiple enclosures that share the same combined SE and TM. See Requirement R1–12.2–3.
YL	ASCII	Up to 255	Hardware Location Code (see Section 12.3, “Hardware Location Codes,” on page 327)
Z?	ASCII	Up to 255	Product Specific Information: may be keyword oriented and ‘,’ delimited.

13 Dynamic Reconfiguration (DR) Architecture

Dynamic Reconfiguration (DR) is the capability of a system to adapt to changes in the hardware/firmware physical or logical configuration, and to be able to use the new configuration, all without having to turn the platform power off or restart the OS. This section will define the requirements for systems that support DR operations.

13.1 DR Architecture Structure

Figure 11, “DR Architecture Structure,” on page 356 shows the relationship of the DR architecture with LoPAPR and the relationship of the individual DR pieces with the base DR architecture. Each specific DR option (for example, PCI Hot Plug) will have a piece that sits on top of the base DR option. The base DR option is the set of requirements that will be implemented by all DR platforms and that will be utilized by the OS that supports any of the specific DR options. The specific DR options will call out the base DR option requirements as being required. Therefore, in the figure, any specific DR option is really that specific DR option piece plus the base DR option. The base DR option is not a stand-alone option; a platform which supports the base DR option without one or more of the specific DR option pieces that sit on top of it, has not implemented the DR architecture to a level that will provide any DR function to the user. Likewise, a DR entity will meet the requirements of at least one of the specific DR options, or else software is not required to support it as a DR entity. Thus, the base DR option is the common building block and structure upon which all other specific DR options are built.

DR operations can be physical or logical. Currently the only physical DR entities are PCI Hot Plug. That is, the OS only has control over the physical DR operations on PCI IOAs. The current direction for hot plug of other DR entities is to do the physical hot plug (power up/down, control of service indicators, etc.) via the HMC and to bring the entity into usage by an OS via logical DR operations (Logical Resource DR -- LRDR). The PCI Hot Plug DR option can be found in Section 13.6, “PCI Hot Plug DR Option,” on page 372. The Logical Resource Dynamic Reconfiguration option can be found in Section 13.7, “Logical Resource Dynamic Reconfiguration (LRDR),” on page 377. It is expected that as time goes on, the base DR option may be expanded upon by addition of other DR options.

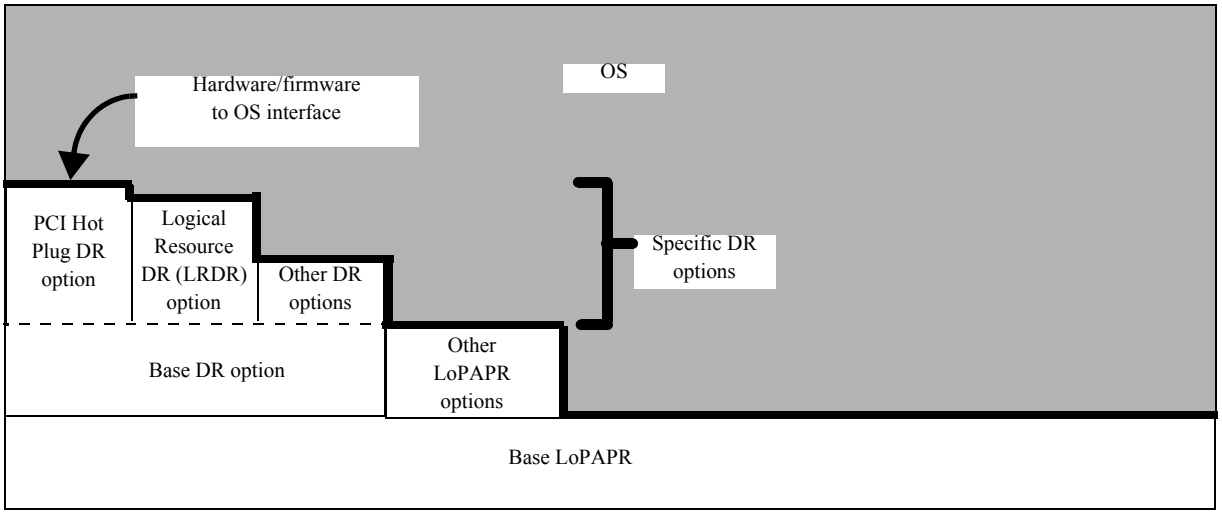


Figure 11. DR Architecture Structure

13.2 Definitions Used in DR

Table 162. DR Definitions

Term	Definition
Base Dynamic Reconfiguration (DR) option	The base on which all of the specific DR options are built. Specific DR options include, for example, the PCI Hot Plug DR option, processor card DR option, etc. These specific DR options each include the requirement that all the base DR option requirements be met. See Section 13.1, “DR Architecture Structure,” on page 355 for more information about the structure of the DR architecture pieces.
Dynamic Reconfiguration (DR)	The capability of a system to adapt to changes in the hardware/firmware configuration with the power on and the OS operating, and to be able to use the new configuration. This is a piece of the High Availability puzzle, but only one of the pieces. Addition, removal, and replacement may, in general, be done with the power on or off on the connector into which the entity is being added or removed. For the PCI Hot Plug option, the power to the slot is turned off and the logic signals are electrically isolated from the connector during the plug or unplug operation.
Depth First	Refers to a method where a tree structure (for example, a set of PCI buses connected by PCI to PCI bridges) is traversed from the top to the bottom before all the siblings at any particular level are acted upon.
DR Connector (DRC)	The term “DR connector” will be used here to define the plug-in point for the entity that is participating in DR. For example, a ‘slot’ into which a PCI IOA is inserted is a DRC.
DR Entity	An entity that can participate in DR operations. That is, an entity that can be added or removed from the platform while the platform power is on and the system remains operational. See also the definitions of logical and physical DR entities.
DR Operation	The act of removing, adding or replacing a DR Entity.
Entity	One or more I/O devices, IOAs, Processor cards, etc., that are treated as one unit.

Table 162. DR Definitions *(Continued)*

Term	Definition
High Availability (HA) System	A system that gives the customer "close" to continuous availability, but allows for some system down-time. Besides DR, other factors that need to be considered in the design of an HA system include system partitioning, clustering, redundancy, error recovery, failure prediction, Error Detection and Fault Isolation (EDFI), software failure detection/recovery, etc.
I/O Adapter (IOA)	A device which attaches to a physical bus which is capable of supporting I/O (a physical IOA) or logical bus (a virtual IOA) and which has its own separate set of resources is referred to as an IOA. The term "IOA" without the usage of the qualifier "physical" or "virtual" will be used to designate a physical IOA. Virtual IOAs are defined further in Chapter 17, "Virtualized Input/Output," on page 597. Resources which must have the capability of being separate (from other devices) include: MMIO Load/Store address spaces, configuration address spaces, DMA address spaces, power domains, error domains, interrupt domains, and reset domains. Note that the hardware of an IOA may allow for separation of these resources but the platform or system implementation may limit the separation (for example, shared error domains). In PCI terms, an IOA may be defined by a unique combination of its assigned bus number and device number, but not including its function number; an IOA may be a single or multi-function device, unless otherwise specified by the context of the text. Examples include LAN and SCSI IOAs. A PCI IOA may exist as multiple device nodes in the OF device tree; that is, the OF may treat separate "functions" in an IOA as separate OF device tree nodes.
IOA: built-in	An IOA that is not pluggable by the user. Sometimes called integrated I/O. As opposed to an IOA that may be removed as part of a plug-in card removal (see definition for a plug-in card, below).
I/O Bus	A hardware interface onto which an IOA can be plugged on a platform. I/O buses discussed here include:
I/O Bus: PCI	The term "PCI" refers to one of: conventional PCI, PCI-X, or PCI Express. The term "bus" in the case of PCI Express refers to a PCI Express link.
I/O Bus: System Bus	The system bus in a platform is normally used only to attach CPUs, memory controllers, and Host Bridges to bridge to I/O buses. A platform's system bus may, in certain circumstances, be used to attach very high speed IOAs. DR of system bus-attached entities is not considered here.
I/O Device	An entity that is connected to an IOA (usually through a cable). A SCSI-attached DASD device is an example. Some I/O devices and their connection points to the IOAs are designed to be plugged while the connection point is operational to the other I/O devices connected to the same IOA, and some are not. For example, while the SCSI bus was not initially designed to have devices added and removed while the SCSI bus was operational, different vendors have found ways to do so. For example, SCSI-attached DASD is pluggable and unpluggable from the SCSI bus in some platforms.
Live Insertion	A DR operation where the power remains on at the DR connector. Live insertion entities are always powered unless the machine power is shut off or unless a subsystem containing those entities is shut off.
Logical DR entity	A DR entity which does not have to be physically plugged or unplugged during a DR operation on that entity. See Table 240, "Currently Defined DR Connector Types," on page 671 for a list of the supported Logical DR types.
Logical Resource DR	The name of the option for support of DR of logical entities. See Section 13.7, "Logical Resource Dynamic Reconfiguration (LRDR)," on page 377.
PCI Hot Plug	DR for PCI plug-in cards where there is a separate power domain for each PCI Hot Plug slot. Platforms which do not provide individual control of power and isolation for each PCI slot but which do provide power and isolation control for groups of PCI slots (that is, multiple slots per power domain), do not provide "PCI Hot Plug," but can support PCI DR.
Physical DR entity	A DR entity which may need to be physically plugged or unplugged during a DR operation on that entity. See Table 240, "Currently Defined DR Connector Types," on page 671 for a list of the supported physical DR types.
Plug-in card	A card which can be plugged into an I/O connector in a platform and which contains one or more IOAs and potentially one or more I/O bridges or switches.
Subsystem	One or more I/O devices, IOAs, Processor cards, etc., that are treated as one unit, for purposes of removal/insertion.

13.3 Architectural Limitations

The DR architecture places a few limitations on the implementations. Current architectural limitations include:

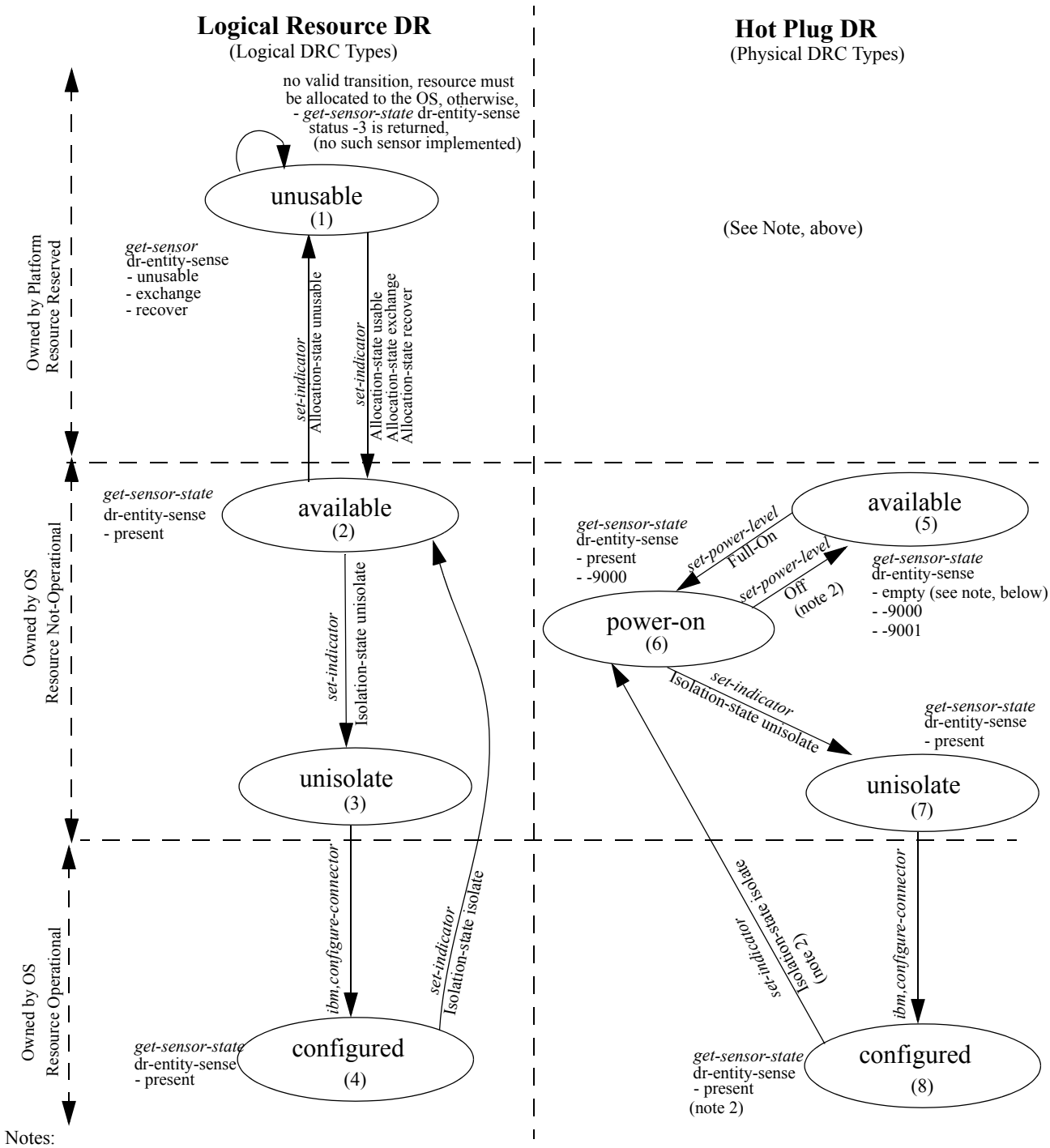
- DR operations will be user initiated at the software level before any physical plugging or unplugging of hardware is performed. This architecture will be flexible enough to add additional methods for invoking the process in the future, but for the initial architecture it will be assumed that the operation is invoked by the user via a software method (for example, invoking an OS DR services program). It is expected that some technologies which will be added in the future will allow plugging/unplugging without the user first informing the software (for example, P1394 and USB).
- Critical system resources cannot be removed via a DR operation. Which system resources are critical will not be defined by this architecture; it is expected that this determination will be made by the OS implementation and/or architecture. Loss of a critical resource would stop the system from operating.
- Many of the RTAS calls will need to work properly, independent of what is powered-off (for example, NVRAM access must work during DR operations). This is partially encompassed by the last bullet. For more information, see Section 13.5.1, “For All DR Options - Platform Requirements,” on page 360.
- Any special requirements relative to redundant power supplies or cooling are not addressed here.
- Moving of a DR entity from one location to another in a platform is supported through a “remove and add” methodology rather than a specific architecture which defines the constructs necessary to allow moving of pieces of the platform around.

Note: The current AIX implementation does a “remove and add” sequence even when the overall DR operation is a replacement. That is, first the old entity is removed, and then the new entity is added.

13.4 Dynamic Reconfiguration State Transitions

Figure 12, “Dynamic Reconfiguration State Transition Diagrams,” on page 359 shows the states and transitions for the dynamic reconfiguration entities (DR Entities). The transition between states is initiated by a program action (RTAS functions) provided the conditions for the transition are met.

Note: Relative to Figure 12, “Dynamic Reconfiguration State Transition Diagrams,” on page 359, physical DRC types are brought in to the “owned by the OS” states either: (1) by the Device Tree at boot time, or (2) by a DLPAR operation, which brings in the logical DRC “above” the physical DRC first, and drags the physical in as part of transferring from state 3 to state 4. Therefore no states appear in the “owned by platform” section under Hot Plug DR in the figure. So, for example, the DLPAR assignment of a PCI physical slot to an OS is done by assigning the logical SLOT DRC above the physical PCI slot, thus giving the following state transitions: state 1, to state 2, to state 3, to state 4, at which time the OS sees the physical slot, sees an IOA in the physical slot (via *get-sensor-state* (dr-entity-sense) of the physical DRC returning “present”), and then proceeds with the state transitions of: state 5, to state 6, to state 7, to state 8. The reverse of this (DLPAR removal of the PCI slot) is: state 8, to state 6, to state 5, to state 4, to state 2, to state 1.



Notes:

1. In State 5, if empty status is returned from the get-sensor-state dr-entity-sense call, then do not attempt to power-on
2. Transitions from State 8 to 6 or from State 6 to 5 may fail (*set-indicator isolation-state* isolate, and *get-sensor-state* dr-entity-sense) if the hardware cannot be accessed to control these operations. In this case, the OS may ignore those errors if the operation is a DLPAR to remove the hardware. See also the **"ibm,ignore-hp-po-fails-for-dlpar"** property in Section B.6.2.1, "Root Node Properties," on page 673.

Figure 12. Dynamic Reconfiguration State Transition Diagrams

13.5 Base DR Option

13.5.1 For All DR Options - Platform Requirements

This section contains the extra requirements placed on the platform for all of the various DR configurations.

At this time, there are no provisions made in the DR architecture for unexpected removal of hardware or insertion of hardware into a DR connector. Therefore the user is expected to interact with the DR software prior to changing the hardware configuration. For example, it is expected that most systems will require a keyboard action prior to the hardware configuration change. Future architecture might allow for other possibilities. For example, a push-button switch at the DR connector may be provided which causes an interrupt to the OS to signal that an operation is about to take place on the connector[1].

As mentioned in Section 13.1, “DR Architecture Structure,” on page 355, the requirements in this section are not stand-alone requirements; the platform will also need to implement one or more of the specific DR options.

R1–13.5.1–1. For all DR options: If the **`“ibm,configure-connector”`** property exists in the **`/rtas`** node of the OF device tree, then the platform must meet all of the requirements for the Base DR option (that is, all of the requirements labeled “For all DR options”), and must also meet all the requirements for at least one of the specific DR options.

R1–13.5.1–2. For all DR options: The platform and OS must adhere to the design and usage restrictions on RTAS routines defined in Table 163, “RTAS Call Operation During DR Operations,” on page 360, and any RTAS calls not specified in Table 163, “RTAS Call Operation During DR Operations,” on page 360 must comply with Table 163 Note 1 and 2.

Table 163. RTAS Call Operation During DR Operations

RTAS Call Name	Reference to Table 163 Note Numbers	*RTAS Call Name*	Reference to Table 163 Note Numbers
rtas-last-error	1	*ibm,read-pci-config*	4
check-exception	1, 2	*ibm,write-pci-config*	4,7
display-character	1	*restart-rtas*	1
event-scan	1, 2	*set-indicator*	3, 4, 5
query-cpu-stopped-state	4	*set-power-level*	3, 4, 5
get-power-level	4	*set-time-for-power-on*	1
get-sensor-state	3, 4	*set-time-of-day*	1
get-time-of-day	1	*start-cpu*	4
ibm,configure-connector	7	*stop-self*	7
ibm,exti2c	1	*system-reboot*	1
ibm,os-term	1	*nvram-store*	1
nvram-fetch	1	*power-off*	1, 6

1.The push-button method is one that has been mentioned as a possible enhancement for systems that are produced for telephone company applications.

Table 163. RTAS Call Operation During DR Operations ***(Continued)***

RTAS Call Name	Reference to Table 163 Note Numbers	*RTAS Call Name*	Reference to Table 163 Note Numbers
ibm,power-off-ups			

Table 163 Notes:

1. These RTAS calls function as specified in this architecture, regardless of the power state of any DR entity in the platform (providing the call is implemented).
2. These RTAS calls do not cause errors nor return an error status by accessing hardware which is isolated, unusable and/or powered down.
3. These RTAS calls function properly when dealing with a DR connector, when the parent of that DR connector is powered and configured, regardless of the state of the child of the parent (for set-indicator, the isolation-state and dr-indicator names, and for get-sensor-state, the dr-entity-sense sensor name).
4. The results of the OS issuing these RTAS calls to hardware when the access to that hardware is through hardware which is isolated, unusable, powered off, or incompletely configured, are indeterminate.
5. The results of the OS changing the power or isolation state of a Dynamic Reconfigure connector while there is an uncompleted *ibm,configure-connector* operation in progress against that connector are indeterminate.
6. Power domains which were defined within sub-trees which have been subsequently isolated may remain un-modified by this call; their state will be platform dependent.
7. The results of the OS issuing these RTAS calls to hardware which is isolated and/or powered off are indeterminate.

R1–13.5.1–3. For all DR options: If there is Forth code associated with a DR entity, it must not modify the OF device tree properties or methods unless modifications can be hidden by the *ibm,configure-connector* RTAS call (that is, where this RTAS routine recognizes the entity and creates the appropriate OF device tree characteristics that would have been created by the Forth code).

R1–13.5.1–4. For all DR options: The hardware must protect against any physical damage to components if the DR entity is removed or inserted while power is on at the DR connector.

R1–13.5.1–5. For all DR options: During a DR operation (including resetting and removing the reset from the entity, powering up and powering down the entity, unisolating and isolating the entity, and physically inserting and removing the entity), the platform must prevent the introduction of unrecoverable errors on the bus or interconnect into which the DR entity is being inserted or removed.

R1–13.5.1–6. For all DR options: During a DR operation (including resetting and removing the reset from the entity, powering up and powering down the entity, unisolating and isolating the entity, and physically inserting and removing the entity), the platform must prevent damage to the DR entity and the planar due to any electrical transitions.

R1–13.5.1–7. For all DR options: If there are any live insertion DR entities in a platform and if those entities or the rest of the platform cannot tolerate the power being turned off to those entities during DR operations on other DR entities, then they must not be placed in the same power domain as the DR entities that will be powered off.

R1–13.5.1–8. For all DR options: A separate visual indicator must be provided for each physical DR connector which can be used for insertion of a DR Entity or which contains a DR entity that can be removed, and the indicator must be individually controllable via the *set-indicator* RTAS call, and must have the capability to be set to the states as indicated in Table 168, “set-indicator Defined Indicators for all DR Options,” on page 368 and Table 171, “Visual Indicator Usage,” on page 371.

R1–13.5.1–9. For all DR options: If a platform provides a separate indicator to indicate the state of the power for the DR connector, then that LED must be turned on by the platform when the platform turns the power on to the DR connector and must be turned off by the platform when the platform turns the power off to the DR connector.

R1–13.5.1–10. For all DR options: If a DR entity requires power to be turned off prior to the physical removal of the DR entity from the platform, then the hardware must provide a green power indicator to indicate the power state of the DR entity

R1–13.5.1–11. For all DR options: The platform must provide any necessary power sequencing between voltages within a power domain during DR operations (for example, during the *set-power-level* RTAS call).

R1–13.5.1–12. For all DR options: If a platform supports DR, then all DR entities must support the full on to off and the off to full on power transitions.

Architecture Note: Requirement R1–13.5.1–2 is necessary so that the OS can count on the availability of certain RTAS facilities and so that the OS does not use other RTAS facilities when they are not available. This may put certain hardware restrictions on what can and cannot be shut down.

Hardware Implementation Notes:

- Requirement R1–13.5.1–2 requires careful planning of hardware design and platform structure to assure that no resources critical to RTAS are put into power domains that are powered down as part of a DR operation. In addition, the platform is required to provide the facilities (registers and bits in registers readable by firmware, etc.) so that RTAS can query the state of the hardware and determine if something is powered off before actually accessing the powered-off hardware.
- Requirement R1–13.5.1–8 indicates that there cannot be any sharing of indicators between DR connectors.
- In some large systems (for example, systems with many racks of equipment) it may not be possible or convenient to view the individual DR visual indicators without opening cabinet doors, etc. In such cases, the designers of such systems could consider putting a "summary" visual indicator where the user could readily see it, which is basically a logical "or" of the visual indicators which are out of sight. For example, in a rack-based system, the drawers might have an indicator on the front of the drawer that indicates if any indicators on the back of the drawer are flashing. This summary indicator will not be accessed by the software (that is, will be transparent to the software) but it is permissible for the indicator to have firmware dependencies.

13.5.2 For All DR Options - OF Requirements

This section describes the OF properties added for DR and any additional requirements placed on OF due to DR.

This section defines a number of new DR properties which are arrays. All properties for a specific DR connector under a node are at the same offset into each array. Also, when the descriptive text states "the first connector" this does not imply any physical position or numbering, but rather a logical "first" connector beneath a particular node in the OF device tree.

13.5.2.1 General Requirements

R1–13.5.2.1–1. For all DR options: When the firmware passes control to the OS, the DR hardware must be initialized such that all of the DR connectors which would return "DR entity present" to a *get-sensor-state* (dr-entity-sense) are fully powered and operational and any DR visual indicators are set to the appropriate state (on or off) as indicated by Table 171, "Visual Indicator Usage," on page 371.

R1–13.5.2.1–2. For all DR options: After the firmware has passed control to the OS, the state of the DR visual indicators must not change except under the following conditions:

- As directed to do so by the *set-indicator* RTAS call.
- Under the condition of a power-fault, in which case the hardware may change the state of the visual indicator to the "off" state if it turns the power off to the slot.

R1–13.5.2.1–3. For all DR options: The platforms which have hierarchical power domains must provide the **"power-domains-tree"** property in the OF device tree.

13.5.2.2 "ibm,drc-indexes" Property

This property is added for the DR option to specify for each DR connector an index to be passed between the OS and RTAS to identify the DR connector to be operated upon. This property is in the parent node of the DR connector to which the property applies. See Section B.6.1, "Properties for Dynamic Reconfiguration," on page 670 for the definition of this property.

R1–13.5.2.2–1. For all DR options: For each OF device tree node which supports DR operations on its children, the OF must provide an **"ibm,drc-indexes"** property for that node.

13.5.2.3 "ibm,my-drc-index" Property

This property is added for the DR option to specify for each node which has a DR connector between it and its parent, the value of the entry in the **"ibm,drc-indexes"** property for that connector. This property is used for correlation purposes. See Section B.6.1, "Properties for Dynamic Reconfiguration," on page 670 for the definition of this property.

R1–13.5.2.3–1. For all DR options: For each OF device tree node which has a DR connector between it and its parent, the OF must provide an **"ibm,my-drc-index"** property for that node.

13.5.2.4 "ibm,drc-names" Property

This property is added for the DR option to specify for each DR connector a user-readable location code for the connector. See Section B.6.1, "Properties for Dynamic Reconfiguration," on page 670 for the definition of this property.

R1–13.5.2.4–1. For all DR options: For each OF device tree node which supports DR operations on its children, the OF must provide an **"ibm,drc-names"** property for that node.

R1–13.5.2.4–2. For all DR options: The content of the **"ibm,drc-names"** property must be of the format defined in Table 164, ""ibm,drc-names" Property Format," on page 363.

Table 164. **"ibm,drc-names"** Property Format

DRC Type	DRC Name
1-8, 11-30 (PCI Hot Plug)	Location code
SLOT	Location code (built-in has port suffix)
PORT	Port x
CPU	CPU x where "x" is a decimal number with one or more digits and no leading zeroes
MEM or MEM-n	LMB x where "x" is a decimal number with one or more digits and no leading zeroes

Table 164. **"ibm,drc-names"** Property Format

DRC Type	DRC Name
PHB	PHB x where "x" is a decimal number with one or more digits and no leading zeroes

13.5.2.5 "ibm,drc-power-domains" Property

This property is added for the DR option to specify for each DR connector the power domain in which the connector resides. See Section B.6.1, "Properties for Dynamic Reconfiguration," on page 670 for the definition of this property.

R1–13.5.2.5–1. For all DR options: For each OF device tree node which supports DR operations on its children, the OF must provide an **"ibm,drc-power-domains"** property for that node.

Software Implementation Notes:

- Software will not call the *set-power-level* RTAS call with an invalid power domain number, and for purposes of this call, a power domain number of -1 (a live insert connector) is considered invalid.
- For the case where the power domain is -1 (the live insert case), this does not imply that the connector does not need isolating before the DR operation, only that it does not need to be powered off.

13.5.2.6 "ibm,drc-types" Property

This property is added for the DR option to specify for each DR connector a user-readable connector type for the connector. See Section B.6.1, "Properties for Dynamic Reconfiguration," on page 670 for the definition of this property.

Architecture Note: The logical connectors (CPU, MEM etc.) represent DR boundaries that may not have physical DR connectors associated with them. If a physical DR boundaries were present they would be represented by a different DR connector type. It is possible that a given boundary may be represented by both a physical and a logical connector. In that case, logical assignment would be managed with the logical connector and physical add/remove would be managed by specifying the physical DR connector.

R1–13.5.2.6–1. For all DR options: For each OF device tree node which supports DR operations on its children, the OF must provide an **"ibm,drc-types"** property for that node.

13.5.2.7 "ibm,phandle" Property

This property is added for the DR option to specify the phandle for each OF device tree node returned by *ibm,configure-connector*. See Section B.6.1, "Properties for Dynamic Reconfiguration," on page 670 for the definition of this property.

R1–13.5.2.7–1. For all DR options: The *ibm,configure-connector* RTAS call will include the **"ibm,phandle"** property in each OF device tree node that it returns. This phandle must be unique and consistent with any phandle visible to an OF client program or any other information returned by *ibm,configure-connector*.

13.5.3 For All DR Options - RTAS Requirements

For platforms that implement DR, there is one new RTAS call and some changes (new requirements) placed on existing ones.

13.5.3.1 General Requirements

The following are the general requirements for RTAS for all DR options.

R1–13.5.3.1–1. For all DR options: If there is Forth code associated with a DR entity and that Forth code would normally modify the OF device tree properties or methods, then if that entity is to be supported as a DR entity on a particular platform, the *ibm,configure-connector* RTAS call on that platform must recognize that entity and create the appropriate OF device tree characteristics that would have been created by the Forth code.

13.5.3.2 *set-power-level*

This RTAS call is defined in Section 7.3.6.1, “set-power-level,” on page 151. Several additional requirements are placed on this call when the platform implements DR along with PM.

This RTAS call is used in DR to power up or power down a DR connector, if necessary (that is, if there is a non-zero power domain listed for the DR connector in the **`“ibm,drc-power-domains”`** property). The input is the power domain and the output is the power level that is actually to be set for that domain; for purposes of DR, only two of the current power levels are of interest: “full on” and “off.”

For sequencing requirements between this RTAS routine and others, see Requirements R1–13.5.4.2–2 and R1–13.5.4.2–3.

R1–13.5.3.2–1. For all DR options: the *set-power-level* RTAS call must be implemented as specified in Section 7.3.6.1, “set-power-level,” on page 151 and the further requirements of this DR option.

R1–13.5.3.2–2. For all DR options: The *set-power-level* RTAS call must initiate the operation and return “busy” status for each call until the operation is actually complete.

R1–13.5.3.2–3. For all DR options: If a DR operation involves the user inserting a DR entity, then if the firmware can determine that the inserted entity would cause a system disturbance, then the *set-power-level* RTAS call must not power up the entity and must return an error status which is unique to that particular type of error, as indicated in Table 165, “set-power-level Error Status for specific DR options,” on page 365.

Table 165. *set-power-level* Error Status for specific DR options

Parameter Type	Name	Option Name	Values
Out	Status	PCI Hot Plug DR option	-9000: Powering entity would create change of frequency on the bus and would disturb the operation of other PCI IOAs on the bus, therefore entity not powered up.

Hardware Implementation Notes:

- For any DR operation, the firmware could optionally not allow powering up of a DR entity, if the powering up would cause a platform over-power condition (the firmware would have to be provided with the DR Entities’ power requirements and the platform’s power capability by a method which is not architected by the DR architecture).
- If PM is not implemented in the platform, then only the “full on” and “off” states need to be implemented for DR and only those two states will be used.

Software Implementation Note: The operation of the *set-power-level* call is not complete at the time of the return from the call if the "busy" status is returned. If it is necessary to know when the operation is complete, the routine should be called with the same parameters until a non-busy status is returned.

13.5.3.3 *get-sensor-state*

This RTAS call is defined in Section 13.5.3.3, "get-sensor-state," on page 366. This RTAS call will be used in DR to determine if there is something connected to the DR connector.

The **"rtas-sensors"** and **"ibm,sensor-**<token>**"** OF properties are not applicable to DR sensors defined in Table 166, "get-sensor-state Defined Sensors for All DR Options," on page 366.

R1–13.5.3.3–1. For all DR options: RTAS must implement the *get-sensor-state* RTAS call.

R1–13.5.3.3–2. For all DR options: The sensor values specified in Table 166, "get-sensor-state Defined Sensors for All DR Options," on page 366 must be implemented as specified in that table.

Table 166. *get-sensor-state* Defined Sensors for All DR Options

Sensor Name	Token Value	Defined Sensor Values	Description
dr-entity-sense	9003	DR connector empty (0)	Returned for physical DR entities if the connector is available (empty) for an add operation. The DR connector must be allocated to the OS to return this value, otherwise a status of -3, no such sensor implemented, will be returned from the *get-sensor-state* RTAS call.
		DR entity present (1)	Returned for logical and physical DR entities when the DR connector is allocated to the OS and the DR entity is present. For physical DR entities, this indicates that the DR connector actually has a DR entity plugged into it. For DR connectors of physical DR entities, the DR connector must be allocated to the OS to return this value, otherwise a status of -3, no such sensor implemented, will be returned from the *get-sensor-state* RTAS call. For DR connectors of logical DR entities, the DR connector must be allocated to the OS to return this value, otherwise a sensor value of 2 or 3 will be returned.
		DR entity unusable (2)	Returned for logical DR entities when the DR entity is not currently available to the OS, but may possibly be made available to the OS by calling *set-indicator* with the allocation-state indicator, setting that indicator to usable.
		DR entity available for exchange (3)	Returned for logical DR entities when the DR entity is available for exchange in a sparing type operation, in which case the OS can claim that resource by doing a *set-indicator* RTAS call with allocation-state set to exchange.
		DR entity available for recovery (4)	Returned for logical DR entities when the DR entity can be recovered by the platform and used by the partition performing a *set-indicator* RTAS call with allocation-state set to recover.

R1–13.5.3.3–3. For all DR options except the PCI Hot Plug and LRDR options: If the *get-sensor-state* call with the dr-entity-sense sensor requires the DR entity to be powered up and/or unisolated to sense the presence of the DR entity, then the *get-sensor-state* call must return the error code of -9000 or -9001, as defined in Table 167, "get-sensor-state Error Status for All DR Options," on page 367, if the DR entity is powered down or is isolated when the call is made.

Table 167. *get-sensor-state* Error Status for All DR Options

Parameter Type	Name	Values
Out	Status	-9000: Need DR entity to be powered up and unisolated before RTAS call -9001: Need DR entity to be powered up, but not unisolated, before RTAS call -9002: (see architecture note, directly below)

Architecture Note: The -9002 return code should not be implemented. For legacy implementations if it is returned, then it should be treated by the caller the same as a return value of 2 (DR entity unusable).

R1–13.5.3.3–4. For all DR options: The value used for the sensor-index input to the g*et-sensor-state* RTAS call for the sensors in Table 166, “get-sensor-state Defined Sensors for All DR Options,” on page 366 must be the index for the connector, as passed in the **`“ibm,drc-indexes”`** property.

Hardware and Software Implementation Note: The status introduced in Requirement R1–13.5.3.3–3 is not valid for *get-sensor-state* calls when trying to sense insertion status for PCI slots (see Requirement R1–13.6.1–5).

Architecture Note: DR entity available for recovery state is intended to allow a platform to temporary allocate to itself resources on a reboot and then allow the OS to subsequently recover those resources when no longer needed by the platform. An example of use would be the platform temporarily reserving some LMBs to itself during a reboot to store dump data, and then making the LMBs available to a OS partition by marking them with the state of “available for recovery” after the dump data has been transferred to the OS.

13.5.3.4 *set-indicator*

This RTAS call is defined as shown in Table 39, “set-indicator Argument Call Buffer,” on page 142. This RTAS call is used in DR to transition between isolation states, allocation states, and control DR indicators. In some cases, a state transition fails due to various conditions, however, a null transition (commanding that the new state be what it already is) always succeeds. As a consequence, this RTAS call is used in all DR sequences to logically (and if necessary physically) isolate and unisolate the connection between a DR entity and the platform. If physical isolation is indeed required for the DR entity, this RTAS call determines the necessity for isolation, not the calling program.

The *set-indicator* allocation-state and *set-indicator* isolation-state are linked. Before calling *set-indicator* with isolation-state set to unisolate, the DR entity being unisolated will first need to be allocated to the OS. If the *get-sensor-state* call would return a value of DR entity unusable or if it would return an error like -3 for the DR entity, then the *set-indicator* isolation-state to unisolate would fail for that DR entity.

For sequencing requirements between this RTAS routine and others, see Requirements R1–13.5.4.2–2 and R1–13.5.4.2–3.

A single *set-indicator* operation for indicator type 9001 may require an extended period of time for execution. Following the initiation of the hardware operation, if the *set-indicator* call returns prior to successful completion of the operation, the call will return either a status code of -2 or 990x. A status code of -2 indicates that RTAS may be capable of doing useful processing immediately. A status code of 990x indicates that the platform requires an extended period of time, and hints at how much time will be required before completion status can be obtained. Neither the 990x nor the -2 status codes imply that the platform has initiated the operation, but it is expected that the 990x status would only be used if the operation had been initiated.

The following are the requirements for the base DR option. Other DR options may put additional requirements on this RTAS call.

Table 168, “set-indicator Defined Indicators for all DR Options,” on page 368 indicates which DR indicators are used with which DR connector types.

The **"rtas-indicators"** and **"ibm,indicator-**<token>**"** OF properties are not applicable to DR indicators defined in Table 168, "set-indicator Defined Indicators for all DR Options," on page 368.

R1–13.5.3.4–1. For all DR options: The indicator state values specified in Table 168, "set-indicator Defined Indicators for all DR Options," on page 368 must be implemented as specified in that table.

Table 168. *set-indicator* Defined Indicators for all DR Options

Indicator Name	Token Value	Defined State Values	Default Value	Examples/Comments
isolation-state	9001	Isolate (0), Unisolate (1)	Unisolated	This indicator must be implemented for DR connectors for both physical and logical DR entities. Isolate refers to the DR action to logically disconnect the DR entity from the platform. An isolate operation makes the DR entity available to the firmware, and in the case of a physical DR entity like a PCI IOA, logically disconnects the DR entity from the platform (for example, from the PCI bus). Unisolate refers to the DR action to logically connect the entity. Before *set-indicator* isolation-state to unisolate, the DR entity being unisolated must first be allocated to the OS. If the *get-sensor-state* call with the dr-entity-sense token would return a value of DR entity unusable or if it would return an error like -3 for the DR entity, then the *set-indicator* isolation-state to unisolate must fail for that DR entity.
dr-indicator	9002	Inactive (0), Active (1), Identify (2) Action (3)	0 if Inactive 1 if Active	This indicator must be implemented for DR connectors for physical DR entities. If the DR indicators exist for the DR connector, then they are used to indicate the state of the DR connector to the user. Usage of these states are as defined in Table 171, "Visual Indicator Usage," on page 371 and Chapter 16, "Service Indicators," on page 511.
allocation-state	9003	unusable (0) usable (1) exchange (2) recover (3)	NA	This indicator must be implemented for DR connectors for logical DR entities. Used to allocate and deallocate entities to the OS. The initial allocation state of a connector is established based upon the initial allocation of resources to the OS image. Subsequently, an OS may request a change of allocation state by use of the *set-indicator* with allocation-state token. If the transition to the usable state is not possible the -3 (no such indicator implemented) status is returned.

R1–13.5.3.4–2. For all DR options: The value used for the indicator-index input to the *set-indicator* RTAS call for the indicators in Table 168, "set-indicator Defined Indicators for all DR Options," on page 368 must be the index for the connector, as passed in the **"ibm,drc-indexes"** property.

R1–13.5.3.4–3. For all DR options: The *set-indicator* call must return a -2 status, or optionally for indicator type 9001 the 990x status, for each call until the operation is complete; where the 990x status is defined in Table 20, "RTAS Status Word Values," on page 119.

R1–13.5.3.4–4. For all DR options: If this is a DR operation that involves the user inserting a DR entity, then if the firmware can determine that the inserted entity would cause a system disturbance, then the *set-indicator* RTAS call must not unisolate the entity and must return an error status which is unique to the particular error.

R1–13.5.3.4–5. For all DR options: If the *set-indicator* index refers to a connector that would return a "DR entity unusable" status (2) to the *get-sensor* dr-entity-sense token, the *set-indicator* RTAS return code must be "No such indicator implemented" (-3), except in response to a successful *set-indicator* allocation state usable.

R1–13.5.3.4–6. For all DR options combined with the LPAR option: The RTAS *set-indicator* specifying unusable allocation-state of a DR connector must unmap the resource from the partition's Page Frame Table(s) and, as appropriate, its Translation Control Entry tables.

R1–13.5.3.4–7. For all DR options combined with the LPAR option: The successful completion of the RTAS *set-indicator* specifying usable allocation-state of a DR connector must allow subsequent mapping of the resource as appropriate within the partition's Page Frame Table(s) and/or its Translation Control Entry tables.

Software Implementation Note: The operation of the *set-indicator* call is not complete at the time of the return from the call if the "busy" status is returned. If it is necessary to know when the operation is complete, the routine should be called with the same parameters until a non-busy status is returned.

Hardware and Software Implementation Note: The *set-indicator* (isolation-state) call is used to clear RTAS internal tables regarding this device. The *ibm,configure-connector* RTAS routine will need to be called before using the entities below this connector, even if power was never removed from an entity while it was in the isolated state.

13.5.3.5 *ibm,configure-connector* RTAS Call

The RTAS function *ibm,configure-connector* is a new RTAS call introduced by DR and is used to configure a DR entity after it has been added by either an add or replace operation. It is expected that the *ibm,configure-connector* RTAS routine will have to be called several times to complete the configuration of a dynamic reconfiguration connector, due to the time required to complete the entire configuration process. The work area contains the intermediate state that RTAS needs to retain between calls. The work area consists of 4096 byte pages of real storage on 4096 byte boundaries which can be increased by one page on each call. The OS may interleave calls to *ibm,configure-connector* for different dynamic reconfiguration connectors, however, a separate work area will be associated with each dynamic reconfiguration connector which is actively being configured. Other standard RTAS locking rules apply.

The properties generated by the *ibm,configure-connector* call are dependent on the type of DR entities. For a list of properties generated, see the RTAS Requirements section for each specific DR option. For example, for a list of properties generated for PCI Hot Plug, see Section 13.6.3, "PCI Hot Plug DR - Run Time Firmware Requirements," on page 374.

For sequencing requirements between this RTAS routine and others, see Requirement R1–13.5.4.2–2.

R1–13.5.3.5–1. For all DR options: The RTAS function *ibm,configure-connector* must be implemented and must implement the argument call buffer defined by Table 169, "ibm,configure-connector Argument Call Buffer," on page 369.

Table 169. *ibm,configure-connector* Argument Call Buffer

Parameter Type	Name	Values
In	*Token*	Token for *ibm,configure-connector*
	Number Inputs	2
	Number Outputs	1
	Work area	Address of work area
	Memory extent	0 or address of additional page
Out	*Status*	-9003: Cannot configure - Logical DR connector unusable, available for exchange, or available for recovery. -9002: Cannot configure - DR Entity cannot be supported in this connector -9001 Cannot configure - DR Entity cannot be supported in this system -2: Call again -1: Hardware error 0: Configuration complete 1: Next sibling 2: Next child 3: Next property 4: Previous parent 5: Need more memory 990X: Extended Delay

R1–13.5.3.5–2. For all DR options: On the first call of a dynamic reconfiguration sequence, the one page work area must be initialized by the OS as in Table 170, “Initial Work Area Initialization,” on page 370.

Table 170. Initial Work Area Initialization

Entry Offset	Value
0	entry from the **“ibm,drc-indexes”** property for the connector to configure
1	0

Architecture Note: The entry offset in Table 170, “Initial Work Area Initialization,” on page 370 is either four bytes or eight bytes depending on whether RTAS was instantiated in 32-bit or 64-bit mode, respectively.

R1–13.5.3.5–3. For all DR options: On all subsequent calls of the sequence, the work area must be returned unmodified from its state at the last return from RTAS.

R1–13.5.3.5–4. For all DR options: The *ibm,configure-connector* RTAS call must update any necessary RTAS configuration state based upon the configuration changes effected through the specified DR connector.

The sequence ends when either RTAS returns a “hardware error” or “configuration complete” status code, at which time the contents of the work area are undefined. If the OS no longer wishes to continue configuring the connector, the OS may recycle the work area and never recall RTAS with that work area. Unless the sequence ends with Configuration Complete, the OS will assume that any reported devices remain unconfigured and unusable. RTAS internal data structures (outside of the work area) are not updated until the call which returns “configuration complete” status. A subsequent sequence of calls to *ibm,configure-connector* with the same entry from the **“ibm,drc-indexes”** property will restart the configuration of devices which were not completely configured.

If the index from **“ibm,drc-indexes”** refers to a connector that would return an “DR entity unusable” status (2) to the *get-sensor* RTAS call with dr-entity-sense token, the *ibm,configure-connector* RTAS call for that index immediately returns “-9003: Cannot configure - Logical DR connector unusable” on the first call without any configuration action taken on the DR connector.

A dynamic reconfiguration connector may attach several sibling OF device tree architected devices. Each such device may be the parent of one or more device sub-trees. The *ibm,configure-connector* RTAS routine configures and reports the entire sub-tree of devices rooted in previously unconfigured architected devices found below the connector whose index is specified in the first entry of the work area, except those that are associated with an empty or unowned dynamic reconfiguration connector; where unowned refers to a DR connector that would return a DR entity unusable, a DR entity available for exchange, or a DR entity available for entity available for recovery value, for a *get-sensor* dr-entity-sense sensor. Configuration proceeds in a depth first order.

If the *ibm,configure-connector* RTAS routine returns with the “call again” or 990x status, configuration is proceeding but had to be suspended to maintain the short execution time requirement of RTAS routines. No results are available. The OS should call the *ibm,configure-connector* RTAS routine passing back the work area unmodified at a later time to continue the configuration process.

If the i*bm,configure-connector* RTAS routine returns with a “Cannot configure - DR Entity cannot be supported in this connector”, then there is a lack of one or more resources at this connector for this DR Entity and there is at least one DR connector in the system into which this DR Entity can be configured. In this case, the DR program should indicate to the user that they need to consult the appropriate system documentation relative to the DR Entity that they are trying to insert into the system.

The “need more memory” status code, is similar in semantics to the “call again” status. However, on the next *ibm,configure-connector* call, the OS will supply, via the *Memory extent* parameter, the address of another page of memory for RTAS to add to the work area in order for configuration to continue. On all other calls to *ibm,configure-connector* the

contents of the *Memory extent* parameter should be 0. It is the responsibility of the OS to recover all work area memory after a sequence of *ibm,configure-connector* calls is completed.

Software Implementation Note: The OS may allocate the work area from contiguous virtual space and pass individual discontiguous real pages to *ibm,configure-connector* as needed.

If the *ibm,configure-connector* RTAS routine returns either the “next sibling” or “next child” status codes, configuration has detected an architected OF device tree device, and is returning its OF device tree node-name. Work Area offset 2 contains an offset within the first page of the work area to a NULL terminated string containing the node-name. Note, if the caller needs to preserve this or any other returned parameters between the various calls of a configuration sequence it will copy the value to its own area. Also, the first call returning configuration data will have a “next child” status code.

The “next property” status code indicates that a subsequent property is being returned for the device. Work Area entry offset 2 contains an offset within the first page of the work area to a NULL terminated string containing the property name. Work Area entry offset 3 contains the length of the property value in bytes. Work Area entry offset 4 contains an offset within the first page of the work area to the value of the property.

Architecture Note: The *ibm,configure-connector* RTAS routine returns those applicable properties that can be determined without interpreting any FCode ROM which is associated with the IOA. Additionally, it is permissible for this RTAS call to be aware of various specific IOAs and emulate the action of any FCode associated with the IOA.

If the *ibm,configure-connector* RTAS routine returns the “previous parent” status code, it has come to the end of the string of siblings, and will back up the tree one level following its depth first order algorithm. The 2nd through 4th work area entries are undefined for this status code.

Software Implementation Notes:

1. Any attempts to configure an already configured connector or one in progress of being configured will produce unpredictable results.
2. The software will put the DR entity in the full on power state before issuing the *ibm,configure-connector* RTAS call to configure the DR entity.

13.5.4 For All DR Options - OS Requirements

13.5.4.1 Visual Indicator States

DR Visual indicator usage will be as indicated in the following requirement, in order to provide for a consistent user interface across platforms. Information on implementation dependent aspects of the DR indicators can be found in Chapter 16, “Service Indicators,” on page 511.

R1–13.5.4.1–1. For all DR options: The visual indicators must be used as defined in Table 171, “Visual Indicator Usage,” on page 371.

Table 171. Visual Indicator Usage

State of indicator	Usage
Inactive	The DR connector is inactive and entity may be removed or added without system disruption. For DR entities that require power off at the connector, then the caller of *set-indicator* must turn power off prior to setting the indicator to this state. See also Chapter 16, “Service Indicators,” on page 511.

Table 171. Visual Indicator Usage

State of indicator	Usage
Identify (Locate)	This indicator state is used to allow the user to identify the physical location of the DR connector. This state may map to the same visual state (for example, blink rate) as the Action state, or may map to a different state. See also Chapter 16, “Service Indicators,” on page 511.
Action	Used to indicate to the user the DR connector on which the user is to perform the current DR operation. This state may map to the same visual state (for example, blink rate) as the Identify state, or may map to a different state. See also Chapter 16, “Service Indicators,” on page 511.
Active	The DR connector is active and entity removal may disrupt system operation. See also Chapter 16, “Service Indicators,” on page 511.

13.5.4.2 Other Requirements

R1–13.5.4.2–1. For all DR options: The OS must detect hierarchical power domains (as specified in the **`“power-domains-tree”`** property) and must handle those properly during a DR operation.

R1–13.5.4.2–2. For all DR options: When bringing a DR entity online, the OS must issue the following RTAS calls in the following order:

a. If the power domain is not 0, then call *set-power-level*

b. *set-indicator* (with the isolation-state token and a state value of unisolate)

c. *ibm,configure-connector*

R1–13.5.4.2–3. For all DR options: When taking a DR entity offline, the OS must issue the following RTAS calls in the following order:

a. *set-indicator* (with the isolation-state token and a state value of isolate)

b. If the power domain is not 0, then call *set-power-level*

R1–13.5.4.2–4. When bringing a DR entity online that utilizes TCEs (see Section 3.2.2.2, “DMA Address Translation and Control via the TCE Mechanism,” on page 65), the OS must initialize the DR entity's TCEs.

13.6 PCI Hot Plug DR Option

This section will develop the requirements over and beyond the base DR option requirements, that are unique to being able to perform DR operations on PCI plug-in cards that do not share power domains with other PCI plug-in cards.

13.6.1 PCI Hot Plug DR - Platform Requirements

A method will be provided to isolate the plug-in card (power and logic signals) and to physically remove the plug-in card from the machine. The physical removal may pose an interesting mechanical challenge, due to the position of the card edge connector relative to the desired direction of insertion of the card from the outside of the machine. In addition, PCI plug-in cards may have internal cables and may span multiple slots. Such mechanical issues are not addressed by this architecture.

This section describes the requirements for the platform when a platform implements the PCI Hot Plug DR option.

R1–13.6.1–1. For the PCI Hot Plug DR option: All platform requirements of the base DR option architecture must be met (Section 13.5.1, “For All DR Options - Platform Requirements,” on page 360).

R1–13.6.1–2. For the PCI Hot Plug DR option: All PCI requirements must be met (for example, timing rules, power slew rates, etc.) as specified in the appropriate PCI specifications, and in the *PCI Standard Hot-Plug Controller and Subsystem Specification [20]*.

R1–13.6.1–3. For the PCI Hot Plug DR option: The hardware must provide two indicators per PCI Hot Plug slot, and all the following must be true:

a. One indicator must be green and the platform must use the indicator to indicate the power state of the PCI Hot Plug slot, turning on the indicator when the slot power is turned on and turning off the indicator when the slot power is turned off.

b. The other indicator must be amber and must be controllable by RTAS, separately from all other indicators, and must be used as a slot Identify indicator, as defined in Chapter 16, "Service Indicators," on page 511.

R1–13.6.1–4. For the PCI Hot Plug DR option: The hardware must provide a separate power domain for each PCI Hot Plug slot, controllable by RTAS, and that power domain must not be used by any other DR connector in the platform.

R1–13.6.1–5. For the PCI Hot Plug DR option: The hardware must provide the capability to RTAS to be able to read the insertion state of each PCI Hot Plug slot individually and must provide the capability of reading this information independent of the power and isolation status of the plug-in card.

R1–13.6.1–6. For the PCI Hot Plug DR option: The hardware must provide individually controllable electrical isolation (disconnect) from the PCI bus for each PCI Hot Plug slot, controllable by RTAS and this isolation when set to the isolation mode must protect against errors being introduced on the bus, and damage to the plug-in cards or planars during the plug-in card power up, power down, insertion, and removal.

R1–13.6.1–7. For the PCI Hot Plug option: A platform must prevent the change in frequency of a bus segment (for example, on the insertion or removal of an plug-in card) while that change of frequency would result in improper operation of the system.

R1–13.6.1–8. For the PCI Hot Plug option: For each PCI Hot Plug slot which will accept only 32-bit (data width) plug-in cards, the platform must:

a. Accommodate plug-in cards requiring up to 64 MB of PCI Memory Space and 64 KB of PCI I/O space

b. For TCE-mapped DMA address space, must provide the capability to map simultaneously and at all times at least 128 MB of PCI Memory space for the slot.

R1–13.6.1–9. For the PCI Hot Plug option: Each PCI Hot Plug slot which will accept 64-bit (data width) plug-in cards, the platform must:

a. Accommodate plug-in cards requiring up to 128 MB of PCI Memory Space and 64 KB of PCI I/O space

b. For TCE-mapped DMA address space, must provide the capability to map simultaneously and at all times at least 256 MB of PCI Memory space for the slot.

R1–13.6.1–10. For the PCI Hot Plug option with PCI Express: The power and isolation controls must be implemented by use of the PCI Standard Hot-Plug Controller (see *PCI Standard Hot-Plug Controller and Subsystem Specification [20]*).

R1–13.6.1–11. For the PCI Hot Plug option with PCI Express: If a PCI Hot Plug DRC contains multiple PEs, then that DRC must be owned by the platform or a trusted platform agent.

Hardware implementation Notes:

1. Surge current protection on the planar is one way to provide the required protection against damage to components if an entity is removed from or inserted into a connector with the power still applied to the connector.

2. Removal of an entity without the proper quiescing operation may result in a system crash.

3. In order for hot plugging of PCI plug-in cards with the system operational to be useful, a mechanical means is needed in order to be able to remove or insert PCI plug-in cards without shutting off system power and without removing the covers above the plug-in cards (which in general, would require powering-down the system).

4. It is recommended that the control of the indicators required by Requirement R1–13.6.1–3 be via the PCI Standard Hot Plug Controller (see *PCI Standard Hot-Plug Controller and Subsystem Specification [20]*).

13.6.2 PCI Hot Plug DR - Boot Time Firmware Requirements

R1–13.6.2–1. For the PCI Hot Plug DR option: All OF requirements of the base DR option architecture must be met (Section 13.5.2, “For All DR Options - OF Requirements,” on page 362).

R1–13.6.2–2. For the PCI Hot Plug DR option: The OF must only generate the **`"clock-frequency"`** OF property for PCI bridge nodes which cannot change bus clock frequency during a PCI Hot Plug operation.

R1–13.6.2–3. For the PCI Hot Plug DR option: The OF must set the PCI configuration register bits and fields appropriately.

Hardware Implementation Note: The OF should leave sufficient gaps in the bus numbers when configuring bridges and switches such that plug-in cards with bridges and switches which are to be supported by the platform’s DR operations can be plugged into every slot in the platform in which those plug-in cards are supported. That is, insertion of an plug-in card that contains a bridge or switch into a platform, requires that there be sufficient available bus numbers allocated to that PCI bus such that new bus numbers can be assigned to the buses generated by the bridges and switches on the plug-in cards.

13.6.3 PCI Hot Plug DR - Run Time Firmware Requirements

R1–13.6.3–1. For the PCI Hot Plug DR option: All RTAS requirements of the base DR option architecture must be met (Section 13.5.3, “For All DR Options - RTAS Requirements,” on page 364).

R1–13.6.3–2. For the PCI Hot Plug DR option: The *set-indicator* RTAS call with a indicator type of isolation-state and a state value of unisolate (1) must not return a “success” status until any IOA on a plug-in card inserted into the PCI slot is ready to accept configuration cycles, and must return a “success” status if the PCI slot is empty.

R1–13.6.3–3. For the PCI Hot Plug DR option: The *ibm,configure-connector* RTAS call must initialize the PCI configuration registers and platform to the same values as at boot time.

Architecture Note: During a DR replace operation, the replacement PCI IOA may not get placed back at the same addresses, etc., as the original DR entity by the firmware (although it has to be placed back into the same DR connector, or it is not a DR replace operation). On a replace operation, the configuration information cannot reliably be read from the IOA being replaced (the IOA might be broken), so the firmware cannot read the configuration information from the old IOA and replace the configuration information into the new IOA.

PCI I/O sub-systems architecturally consist of two classes of devices, bus bridges (Processor Host Bridges (PHBs), PCI to PCI Bridges, and PCI Express switches and bridges) and IOAs. The support that *ibm,configure-connector* provides for these two classes is different.

For Bus Bridges, firmware will totally configure the bridge so that it can probe down the depth of the tree. For this reason, the firmware must include support for all bridges the platform supports. This includes interrupt controllers as well as miscellaneous unarchitected devices that do not appear in the OF device tree. The properties supported and reported are the same as provided by the boot time firmware.

For PCI plug-in cards, the support is significantly less; it is essentially the functionality specified in section 2.5 FCode Evaluation Semantics of the *PCI Bus binding to: IEEE Std 1275-1994, Standard for Boot (Initialization, Configuration) Firmware* [6]. However, the configuration proceeds as if all devices do not have an expansion ROM since the RTAS code does not attempt to determine if an FCode ROM is present nor attempts to execute it. This may, in some cases, generate different device node properties, values and methods than would happen had the IOA been configured during boot. If the IOA's device driver or configuration support cannot deal with such differences, then the IOA is not dynamically reconfigurable. The other properties generated are dependent upon the IOA's configuration header from the following list. If the property is not on this list the reader should assume that RTAS *ibm,configure-connector* will not generate it.

Table 172, "PCI Property Names which will be Generated by ibm,configure-connector," on page 375 shows what PCI OF properties can be expected to be returned from the *ibm,configure-connector* call for PCI Hot Plug operations and Table 173, "Non-exhaustive list of PCI properties that may not be generated by ibm,configure connector," on page 376 shows some which can be expected to not be returned.

R1–13.6.3–4. For the PCI Hot Plug DR option: The *ibm,configure-connector* RTAS call when used for PCI IOAs must return the properties named in Table 172, "PCI Property Names which will be Generated by ibm,configure-connector," on page 375 except as indicated in the Present?/Source column.

Table 172. PCI Property Names which will be Generated by *ibm,configure-connector*

Property Name	Present?/Source
"name"	Always present.
"vendor-id"	Always present. From PCI header.
"device-id"	Always present. From PCI header.
"revision-id"	Always present. From PCI header.
"class-code"	Always present. From PCI header.
"interrupts"	Only present if Interrupt Pin register not 0.
"min-grant"	Present unless Header Type is 0x01.
"max-latency"	Present unless Header Type is 0x01.
"devsel-speed"	Only present for conventional PCI and PCI-X.
"compatible"	Always present. Constructed from the PCI header information for the IOA or bridge.
"fast-back-to-back"	Only present for conventional PCI and PCI-X when Status Register bit 7 is set.
"subsystem-id"	Only present if "Subsystem ID" register not 0.
"subsystem-vendor-id"	Only present if "Subsystem vendor ID" register not 0.
"66mhz-capable"	Only present for conventional PCI and PCI-X when Status Register bit 5 is set.
"133mhz-capable"	Only present for PCI-X when PCI-X Status Register bit 17 is set.
"266mhz-capable"	Only present for PCI-X when PCI-X Status Register bit 30 is set.
"533mhz-capable"	Only present for PCI-X when PCI-X Status Register bit 31 is set.
"reg"	Always present. Specifies address requirements.
"assigned-addresses"	Always present. Specifies address assignment.

Table 172. PCI Property Names which will be Generated by *ibm,configure-connector* ***(Continued)***

Property Name	Present?/Source
"ibm,loc-code"	Always present. RTAS will have to remember the location codes associated with all DR connectors so that it can build this property.
"ibm,my-drc-index"	Always present.
"ibm,vpd"	Always present for sub-systems and for PCI IOAs which follow the PCI VPD proposed standard. See Requirement R1–12.4.2–1 and note to see the effect of using different PCI versions.
"device_type"	For bridges, always present with a value of "PCI" otherwise not present.
"ibm,req#msi"	Present for all PCI Express IOA nodes which are requesting MSI support, when the platform supports MSIs.

Table 173, "Non-exhaustive list of PCI properties that may not be generated by ibm,configure connector," on page 376 is a non-exhaustive list of common properties that may not be generated by RTAS *ibm,configure connector* for a PCI IOA. Also, the concept of a phandle does not apply to nodes reported by *ibm,configure-connector.*

Table 173. Non-exhaustive list of PCI properties that may not be generated by *ibm,configure connector*

Property Name	Present?/Source
"ibm,connector-type"	Never present -- only for built-in entries not for pluggable ones.
"ibm,wrap-plug-pn"	Never present -- only for built-in entries not for pluggable ones.
"alternate-reg"	Never present -- needs FCode.
"fcode-rom-offset"	Never present -- RTAS does not look for this.
"wide"	Never present -- needs FCode.
"model"	Never present -- needs FCode.
"supported-network-types"	Never present -- needs FCode.
"address-bits"	Never present -- needs FCode.
"max-frame-size"	Never present -- needs FCode.
"local-mac-address"	Never present -- needs FCode.
"mac-address"	Never present -- needs FCode.
"built-in"	Not present for a PCI Hot Plug connectors.

Architecture Note: Without **"device_type"** and other properties, the OS cannot append an IOA added via DR to the boot list for use during the next boot.

R1–13.6.3–5. For the PCI Hot Plug option: When *ibm,configure-connector* RTAS call returns to the caller, if the device driver(s) for any IOA(s) configured as part of the call are EEH unaware (that is may produce data integrity exposures due to an EEH stopped state) or if they may be EEH unaware, then the *ibm,configure-connector* call must disable EEH prior to returning to the caller.

Software Implementation Note: To be EEH aware, a device driver does not need to be able to recover from an EEH stopped state, only recognize the all-1's condition and not use data from operations that may have occurred since the last all-1's checkpoint. In addition, the device driver under such failure circumstances needs to turn off

interrupts (using the *ibm,set-int-off* RTAS call) in order to make sure that any (unserviceable) interrupts from the IOA do not affect the system. Note that this is the same device driver support needed to protect against an IOA dying or against a no-DEVSEL type error (which may or may not be the result of an IOA that has died). Note that if all-1's data may be valid, the *ibm,read-slot-reset-state2* RTAS call should be used to discover the true EEH state of the device.

13.6.4 PCI Hot Plug DR - OS Requirements

R1–13.6.4–1. For the PCI Hot Plug DR option: All OS requirements of the base DR option architecture must be met (Section 13.5.4, "For All DR Options - OS Requirements," on page 371).

13.7 Logical Resource Dynamic Reconfiguration (LRDR)

The Logical Resource Dynamic Reconfiguration option allows a platform to make available and recover platform resources such as CPUs, Memory Regions, Processor Host Bridges, and I/O slots to/from its operating OS image(s). The Logical Resource Dynamic Reconfiguration option provides the means for providing capacity on demand to the running OS and provides the capability for the platform to make available spare parts (for example, CPUs) to replace failing ones (called *sparing* operations). Combined with the LPAR option, platforms can move resources between partitions without rebooting the partitions' OS images.

The Logical Resource Dynamic Reconfiguration (LRDR) option deals with logical rather than physical resources. These logical resources are already physically installed (dynamic installation/removal of these resources, if supported, is managed via the Hardware Management Console (HMC) or Service Focal Point (SFP)). As such, the OS does not manage either connector power or DR visual indicators. Logical connector power domains are specified as "hot pluggable" (value -1) and DR visual indicators are not defined for logical connectors.

The device tree contains logical resource DR connectors for the maximum number of resources that the platform can allocate to the specific OS. In some cases such as for processors and PHBs, this may be the maximum number of these resources that the platform supports even if there are fewer than that currently installed. In other cases, such as memory regions in a LPARed system, the number may be limited to the amount of memory that can be supported without resizing the cpu page frame table. The OS may use the *get-sensor-state* RTAS call with the dr-entity-sense token to determine if a given drc-index refers to a connector that is currently usable for DR operations. If the connector is not currently usable the return state is "DR entity unusable" (2). A *set-indicator* (isolation state) RTAS call to an unusable connector or (dr-indicator) to any logical resource connector results in a "No such indicator implemented" return status.

Two allocation models are supported. In the first, resources are specifically assigned to one and only one partition at a time by the HMC. In this model, a DR entity state is changed from unusable to usable only by firmware in response to HMC requests to explicitly move the allocation of the resource between partitions. In the second model, certain resources may "float" between cooperating partitions, a partition issues a *set-indicator* (allocation state usable) RTAS call and if the resource is free, the firmware assigns the resource to the requesting partition and returns the success status. *Set-indicator* returns the code "no-such-indicator" if either the resource is not free, or the platform is operating in the first model. To return a resource to the platform firmware, the OS issues a *set-indicator* (allocation state unusable) RTAS call for the resource's DR connector.

13.7.1 Platform Requirements for LRDR

The following requirements apply to the hardware and/or firmware as a result of implementing LRDR on a platform.

R1–13.7.1–1. For the LRDR option: The hardware must provide the capability to power-cycle any hardware that is going to be switched between partitions as part of LRDR, if that hardware requires power-cycling to put the hardware into a known state (for example, PCI IOAs).

Architecture Note: Except for PCI Express IOAs that implement the Function Level Reset (FLR) option, since the PCI architecture is not specific as to the state of the IOA when the IOAs reset is activated and deactivated, either the platform designer will need to guarantee that all logic in all IOAs (including any internal storage associated with the IOA) is cleared to a known state by use of the IOAs' reset, or else the platform will need to provide the capability to power-cycle those IOAs, including the integrated ones (that is, including the non-pluggable ones). Also note that hardware which requires power-cycling to initialize may impact the capability to reliably reboot an OS, independent of whether or not LRDR is implemented.

R1–13.7.1–2. For the LRDR option: Any power-cycling of the hardware which is done by the platform during an LRDR operation (for example, as part of an *ibm,configure-connector* operation), must be functionally transparent to the software, except that PCI plug-in cards that are plugged into a PCI Hot Plug DR connector do not need to be powered on before the *ibm,configure-connector* call for a logical SLOT DR connector returns to the caller.

Architecture Note: PCI plug-in cards that are plugged into a DR connector will not be configured as part of an *ibm,configure-connector* operation on a logical DR connector of type SLOT above the plug-in card (see section 17.6.3.3 *ibm,configure-connector*). However, Requirement R1–13.7.1–2 does require a PCI IOA which is not plugged in to a PCI Hot Plug DR connector (for example, soldered on the planar) be powered up and configured as a result of an *ibm,configure-connector* operation on a logical DR connector of type SLOT above such an IOA, and requires this powering up to be functionally transparent to the caller of *ibm,configure-connector* operation (a longer busy time is not considered to be a violation of the functional transparency requirement).

13.7.2 DR Properties for Logical Resources

Logical resource dynamic reconfiguration is a special case of general DR, therefore, certain DR properties take on special values.

Table 174. DR Property Values for Logical Resources

Property Name	Property Value
"ibm,drc-indexes"	As defined in Section 13.5.2.2, ""ibm,drc-indexes" Property," on page 363.
"ibm,my-drc-index"	As defined in Section 13.5.2.3, ""ibm,my-drc-index" Property," on page 363.
"ibm,drc-names"	As defined in Section 13.5.2.4, ""ibm,drc-names" Property," on page 363. Note: This name allows for correlation between the OS and HMC user interfaces.
"ibm,drc-power-domains"	Logical Resource connectors are defined to be "hot pluggable" having a domain value of -1 per definition in Section 13.5.2.5, ""ibm,drc-power-domains" Property," on page 364.
"ibm,drc-types"	Shall be one of the values "CPU", "MEM", "PHB", or "SLOT" as defined in Table 240, "Currently Defined DR Connector Types," on page 671.

R1–13.7.2–1. For the LRDR option: All platform requirements of the base DR option architecture must be met (Section 13.5.1, "For All DR Options - Platform Requirements," on page 360).

R1–13.7.2–2. For the LRDR option: The **/cpus** OF device tree node must include **"ibm,drc-types"** (of type CPU)**, "ibm,drc-power-domains"** (of value -1)**, "ibm,drc-names",** and **"ibm,drc-indexes"** properties with entries for each potentially supported dynamically reconfigurable processor.

R1–13.7.2–3. For the LRDR option: The root node of the OF device tree must include **"ibm,drc-types"** (of type MEM), **"ibm,drc-power-domains"** (of value -1), **"ibm,drc-names"**, and **"ibm,drc-indexes"** properties with entries for each potentially supported dynamically reconfigurable memory region.

R1–13.7.2–4. For the LRDR option: The root node of the OF device tree must not include any drc properties (**"ibm,drc-*"**) for the base memory region (reg value 0).

R1–13.7.2–5. For the LRDR option: The root node of the OF device tree must include **"ibm,drc-types"** (of type PHB), **"ibm,drc-power-domains"** (of value -1), **"ibm,drc-names"**, and **"ibm,drc-indexes"** properties with entries for each potentially supported dynamically reconfigurable PHB.

R1–13.7.2–6. For the LRDR option: The **/pci** OF device tree node representing a PHB must include **"ibm,drc-types"** (of type SLOT), **"ibm,drc-power-domains"** (of value -1), **"ibm,drc-names"**, and **"ibm,drc-indexes"** properties with entries for each potentially supported dynamically reconfigurable PCI SLOT.

R1–13.7.2–7. For the LRDR option: platforms must implement the allocation-state indicator 9003, as defined in Table 168, "set-indicator Defined Indicators for all DR Options," on page 368.

R1–13.7.2–8. For the LRDR option: For memory LRDR, the **"ibm,lrdr-capacity"** property must be included in the **/rtas** node of the partition device tree (see Section B.6.3.1, "RTAS Node Properties," on page 690).

13.7.3 Architectural Intent -- Logical DR Sequences:

This architecture is designed to support the logical DR sequences specified in the following sections. See also Section 13.4, "Dynamic Reconfiguration State Transitions," on page 358.

13.7.3.1 Acquire Logical Resource from Resource Pool

1. The OS responds to some stimuli (command, workload manager, HMC, etc.) to acquire the resource, perhaps using the **"ibm,drc-names"** value as a reference if a human interface is involved.
2. The OS determines if the resource is usable:
   a. OS uses *get-sensor-state* (dr-entity-sense) to determine the state of the DR connector
   b. If the state is "unusable" the OS issues *set-indicator* (allocation-state, usable) to attempt to allocate the resource. Similarly, if the state is "available for exchange" the OS issues *set-indicator* (allocation-state, exchange) to attempt to allocate the resource, and if the state is "available for recovery" the OS issues *set-indicator* (allocation-state, recover) to attempt to allocate the resource.
   c. If successful, continue, else return error status to the requester. If successful, this is the point where the resource is allocated to the OS.
3. Continue with DR operation.
   a. The OS unisolates the resource via *set-indicator* (isolation-state, unisolate). This is the point where the OS takes ownership of the resource from the platform firmware and the firmware removes the resource from its resource pool.
   b. The OS configures the resource using *ibm,configure-connector* RTAS.
   c. The OS incorporates the resource into its resource pool.
      1. If the resource is a processor, the OS must use the *start-cpu* RTAS call to move the processor from the stopped state (at the end of the *ibm,configure-connector*) to the running state.

d. The OS returns status of operation to the requester.

4. The OS notifies requesting entity of the OS state relative to the resource acquisition.

13.7.3.2 Release Logical Resource

1. Some entity (System administrator commanding from the HMC, a workload manager, etc.) requests the OS to release the resource using the **"ibm,drc-names"** value as a reference.

   a. The OS attempts to stop using logical resource.

      1. If the resource is a processor, the OS calls the *stop-self* RTAS call then waits for the processor to enter the stopped state using the RTAS *query-cpu-stopped-state* call.

      2. The OS isolates the resource via *set-indicator* (isolation-state, isolate).

      3. Unless the isolated resource was the partition's last processor, the OS deallocates the resource via *set-indicator* (allocation-state, unusable). This is the point where the platform firmware takes ownership of the resource from the OS. That is, the OS removes the resource from its resource pool and the firmware adds it to the firmware resource pool.

   b. The OS returns status of operation to the requester.

2. The OS unallocates the resource by *set-indicator* (allocation-state, unusable).

3. The system administrator may command the HMC to allocate the logical resource to another partition (LPAR) or reserved pool (COD).

4. Any needed hardware removal is handled by HMC/SPC.

13.7.4 RTAS Call Semantics/Restrictions

This section describes the unique application of DR RTAS functions to the dynamic reconfiguration of logical resources.

13.7.4.1 *set-indicator* (isolation-state, isolate)

Dynamic reconfiguration of logical resources introduces special meaning and restrictions to the DR connector isolation function depending upon the logical resource being isolated.

13.7.4.1.1 Isolation of CPUs

The isolation of a CPU, in all cases, is preceded by the *stop-self* RTAS function for all processor threads, and the OS insures that all the CPU's threads are in the RTAS stopped state prior to isolating the CPU. Isolation of a processor that is not stopped produces unpredictable results. The stopping of the last processor thread of a LPAR partition effectively kills the partition, and at that point, ownership of all partition resources reverts to the platform firmware.

R1–13.7.4.1.1–1. For the LRDR option: Prior to issuing the RTAS *set-indicator* specifying isolate isolation-state of a CPU DR connector type, all the CPU threads must be in the RTAS stopped state.

R1–13.7.4.1.1–2. For the LRDR option: Stopping of the last processor thread of a LPAR partition with the *stop-self* RTAS function, must kill the partition, with ownership of all partition resources reverting to the platform firmware.

13.7.4.1.2 Isolation of MEM Regions

Isolation of a MEM region creates a paradox if the MEM region being isolated contains the calling program (there being no program left for the firmware to return).

NOTE: The base memory region (starting at address zero) is not associated with a MEM DR connector. This means that the base memory region cannot be isolated. This restriction avoids two fatal conditions, attempts to isolate the region containing RTAS, and attempts to isolate the region containing the interrupt vectors.

It is the responsibility of the OS to unmap the addresses of the MEM region being isolated from both PFT and the TCE tables. When the LRDR option is combined with the LPAR option, the hypervisor ensures that the addresses of the MEM region being isolated are unmapped from both the PFT and TCE tables before successfully completing the isolation of the MEM region. If any valid mappings are found, the RTAS *set-indicator* (isolation-state) does not change the isolation-state and returns with a *Status* -9001 (Valid outstanding translation).

R1–13.7.4.1.2–1. For the LRDR option: The caller of the RTAS *set-indicator* specifying isolate isolation-state of a MEM DR connector type must not be within the region being isolated.

R1–13.7.4.1.2–2. For the LRDR option combined with the LPAR option: The RTAS *set-indicator* specifying isolate isolation-state of a MEM DR connector type must check that the region is unmapped from both the partition's Page Frame Table(s) and any Translation Control Entries that would reference the memory, else the RTAS routine must return with a status of *Status* -9001 (Valid outstanding translation) and the isolation-state is not changed.

Implementation Note: The algorithm chosen for implementing Requirement R1–13.7.4.1.2–2 depends upon the expected frequency of isolation events. For RAS reasons, they should be seldom. For load balancing, they may be far more frequent. These methods are briefly described here:

- First pull the corresponding logical address from the partition's valid space so setting new translations to the logical address are not possible. Then wait for any current in flight translation additions to complete. Followed by either scanning the entire PFT and TCE tables looking for valid translations or checking a use count for the particular logical address range. The PFT/TCE table search may be long, however, it is only done at isolation time.
- The use count method must be maintained for each add and remove of an address translation with the corresponding accessing of a use count based upon the physical real address of the memory block.

13.7.4.1.3 Isolation of PHBs and Slots

An isolation of a PHB naturally disconnects the OS image from any of the DR connectors downstream of the PHB (specifically any I/O slots and PCI Hot Plug connectors associated with the PHB). To avoid the complexity of gracefully managing multi-level isolation, isolation is restricted to only "leaf" DR connectors, that is connectors that have no unisolated or usable DR connectors below them. That is, for logical DR connectors below the connector being isolated, a *get-sensor-state* dr-entity-sense needs to return an unusable (2) and for physical DR connectors below the connector being isolated, the DR entity needs to be isolated first via *set-indicator* (isolation-state, isolate). The OS is responsible for removing all virtual address mappings to the address range associated with a logical I/O SLOT before making the RTAS *set-indicator* (isolation-state) call that isolates the SLOT. When the LRDR option is combined with the LPAR option, the hypervisor ensures that the addresses associated with the logical SLOT being isolated are unmapped from both the PFT and TCE tables before successfully completing the isolation of the SLOT connector. If any valid mappings are found, the RTAS *set-indicator* (isolation-state) does not change the isolation-state and returns with a *Status* -9001 (Valid outstanding translation).

R1–13.7.4.1.3–1. For all LRDR options: If a request to *set-indicator* (isolation-state, isolate) would result in the isolation of one or more other DR connectors which are currently unisolated or usable, then the *set-indicator* RTAS must fail with a return code of "Multi-level isolation error" (-9000).

R1–13.7.4.1.3–2. For the LRDR option combined with the LPAR option: The RTAS *set-indicator* specifying isolate isolation-state of a SLOT DR connector type must check that the IOA address range associated with the slot is unmapped from both the partition's Page Frame Table(s) and any Translation Control Entries that would reference those locations, else the RTAS routine must return with a *Status* -9001 (Valid outstanding translation) and the isolation-state is not changed.

13.7.4.2 *set-indicator* (dr-indicator)

Logical connectors do not have associated dr-indicators (token value 9002). An attempt to set the state of such an indicator results in a "No such indicator implemented" return status.

R1–13.7.4.2–1. For all LRDR options: The calling of *set-indicator* with a token value of 9002 (dr-indicator) and an index representing a logical connector must fail with a return code of "No such indicator implemented" (-3).

13.7.4.3 ibm,configure-connector

The *ibm,configure-connector* RTAS call is used to return to the OS the device tree nodes and properties associated with the newly un-isolated logical resources and configure them for use.

The *ibm,configure-connector* RTAS call used against a logical DR connector can encounter other logical DR connectors or physical DR connectors below it in the tree. If a logical connector is encountered below a logical connector that is being configured, the *ibm,configure-connector* RTAS call will not configure the sub-tree, if it is not owned by the OS (where owned refers to a DR connector that would return a DR entity usable, for a *get-sensor* dr-entity-sense sensor). If a physical connector is encountered, then the sub-tree below the physical connector may or may not be configured, depending on the implementation.

Architecture Note: The requirements of this section specify the minimum sub-tree contents returned for various connector types. Implementations may optionally return other valid previously reported nodes that represent the current configuration of the device tree. Previously reported nodes may not have any changes from their previously reported state. A node that was removed from the configuration due to a DR operation and returns due to a subsequent DR operation is not considered to have been previously reported. It is the caller's responsibility to recognize previously reported nodes.

R1–13.7.4.3–1. For all LRDR options: If a request to *ibm,configure-connector* specifies a connector that is isolated, *ibm,configure-connector* must immediately return configuration complete.

R1–13.7.4.3–2. For all LRDR options: If the connector index refers to a connector that would return a "DR entity unusable" status (2), "DR entity available for exchange" status (3), or "DR entity available for recovery" status (4) to the *get-sensor* dr-entity-sense token, the *ibm,configure-connector* RTAS call must return "-9003: Cannot configure - Logical DR connector unusable, available for exchange, or available for recovery" on the first call without any configuration action taken on the DR connector.

R1–13.7.4.3–3. For all LRDR options: If a request to *ibm,configure-connector* specifies a connector of type CPU, the returned sub-tree must consist of the specific cpu-node, its children, and any referenced nodes that had not been previously reported (such as L2 and L3 caches etc.) all containing the properties as would be contained in those nodes had they been available at boot time.

Implementation Note: Future platforms that support concurrent maintenance of caches, will require that high level cache nodes (L2, L3 etc.) are added by *ibm,configure-connector* such that their properties can change as new/repaired hardware is added to the platform. Therefore, it is the OS's responsibility when isolating a CPU to purge any information it may have regarding an orphaned high level cache node. The OS may use the **"ibm,phandle"**

property to selectively remove caches when a processor is removed. The platform considers any high level cache that is newly referenced (reference count for this partition goes from 0 to 1) to have been previously unreported.

R1–13.7.4.3–4. For all LRDR options: If a request to *ibm,configure-connector* specifies a connector of type MEM, the returned sub-tree must consist of the specific **`ibm,memory-region`** node containing the properties as would be contained in that node had it been available at boot time.

R1–13.7.4.3–5. For all LRDR options: If a request to *ibm,configure-connector* specifies a connector of type PHB or SLOT, then all of the following must be true:

a. The returned values must represent the sub-tree for the specific I/O sub-system represented by the connector, except for entities below any DR connectors (logical or physical) which are below the connector which is the target of the *ibm,configure-connector* operation (that is, the *ibm,configure-connector* operation stops at any DR connector).

b. The sub-tree must consist of the specific node and its children all containing the properties as would be contained in those nodes had they been available at boot time, including (if they exist) built-in PCI IOAs.

R1–13.7.4.3–6. For all LRDR options: If a request to *ibm,configure-connector* specifies a connector of type SLOT, the returned values must represent the sub-tree for the specific I/O sub-system represented by the SLOT connector, and the sub-tree must consist of the specific **`/pci`** node and its children all containing the properties as would be contained in those nodes had they been available at boot time, except for the PCI IOA nodes assigned to the OS image that contain the same properties as they would following a PCI hot plug operation (see Section 13.6.3, “PCI Hot Plug DR - Run Time Firmware Requirements,” on page 374).

R1–13.7.4.3–7. For all LRDR options: If a platform implementation powers-up and configures physical DR entities in the sub-tree under a logical DR connector, then a request to *ibm,configure-connector* of the logical DR connector must use the return status of 990x from the *ibm,configure-connector* call, as necessary, during the DR entity power-up sequence(s) and must control any power-up and sequencing requirements, as would be done by the platform during platform power-up.

14 Logical Partitioning Option

14.1 Overview

The Logical PARtitioning option (LPAR) simultaneously runs one or more copies of a single OS or multiple heterogeneous LoPAPR compliant OSs on a single LoPAPR platform. A partition, within which an OS image runs, is assigned a non-overlapping sub-set of the platform's resources. These platform-allocatable resources include one or more architecturally distinct processors with their interrupt management area, regions of system memory, and I/O adapter bus slots. Partition firmware loaded into each partition generates an OF device tree to represent the resources of the partition to the OS image. Allocatable resources are directly controlled by an OS. This architecture restricts the sharing of allocatable resources between partitions; to do so requires the use of optional facilities presented in 17.2.1.5, "Shared Logical Resources," on page 605. Platform resources, other than allocatable resources mentioned above, that are represented by OF nodes in the device tree of more than one partition (for example, memory controllers and processor host bridges) are marked '*used-by-rtas*'.

Since one of the main purposes of partitioning is isolation of the OSs, the ability to manage real system resources that are common to all partitions is modified for the LPAR option. This means that partition use of RTAS functions which ostensibly use real system resources such as power and time-of-day clocks are buffered from actual manipulation of those resources. The RTAS is modified for LPAR, and has hypervisor support, to virtualize real resources for the partitions. Operational management of the platform moves to a Hardware Management Console (HMC) which is an application, either local or remote, that manages platform resources with messages to the hypervisor rather than being under direct control of a partition's OS.

Platforms supporting LPAR, contain Power PC processors that support the hypervisor addressing mode, in which the physical address is equal to the effective address and all processor resources are available. The "Real Mode" addressing mode, in these processors, is redefined to translate and limit the physical addresses that the processor can access and to restrict access to certain address translation controlling processor resources. The virtual addressing mode is unchanged. See the *Power ISA* [1] (level 2.0 and beyond) for the architecture extensions required for the processor.

The I/O subsystems of these platforms contain I/O bridges that restrict the bus addresses that I/O adapters can access. These restricted bus addresses are subsequently translated through the Translation Control Entry (TCE) mechanism to restrict Direct Memory Accesses (DMAs) by I/O devices. This restriction is to system memory allocated to a partition and managed by the OS image owning the device. The interrupt subsystem of these platforms is enhanced with multiple (one per partition) global interrupt queues to direct interrupts to any processor assigned to the I/O adapter's owning OS image.

Logical Partitioning platforms employ a unique firmware component called the hypervisor (that runs in hypervisor mode) to manage the address mapping and common platform hardware facilities, thereby ensuring isolation between partitions. The OS uses new hypervisor interfaces to manage the partition's page frame and TCE tables. The platform firmware utilizes implementation dependent interfaces to platform hardware common to all partitions. Thus, a system with LPAR has different OS support than a system without LPAR.

In addition to generating per partition device trees, the OF component of a logically partitioned platform manages the initial booting and any subsequent booting of the specific OS image associated with each partition.

The NVRAM of a platform contains configuration variables, policy options, and working storage that is protected from accesses that might adversely affect one or more partitions and their OS images. The hypervisor firmware component

restricts an OS image's access to NVRAM to a region assigned to its partition. This may restrict the number of partitions.

Most system management on systems without LPAR is performed by OS based applications that are given access to modify the platform's configuration variables, policy options and firmware flash. For various Reliability Availability and Serviceability (RAS) reasons, LoPAPR Logical Partitioning platforms do not restrict platform operational management functions to applications running on a preferred partition or OS image. Access to these Operational Management facilities is provided via a Support Processor communication port that is connected to an HMC and/or through a communications port that is connected through a PCI adapter in a partition. The HMC is a set of applications running in a separate stand-alone platform or in one of the platform's partitions. These HMC applications are required to establish the platform's LPAR configuration, however, the configuration is stored in the platform and, therefore, the HMC is not required to boot or operate the platform in a pre-configured non-error condition.

14.1.1 Real Mode Accesses

When the OS controlling an LPAR runs with address translation turned off (MSR_{DR} or MSR_{IR} bit(s) =0) (real mode) the LPAR hardware translates the memory addresses to an LPAR unique area known as the Real Mode Area (RMA). When control is initially passed to the OS from the platform, the RMA starts at the LPAR's logical address 0 and is the first logical memory block reported in the LPAR's device tree. In general, the RMA is a subset of the LPAR's logical address space. Attempting a non relocated access beyond the bounds of the RMA results in an storage interrupt (ISI/DSI depending upon instruction or data reference). The RMA hardware translation scheme is platform dependent. The options are given below.

14.1.1.1 Offset and Limit Registers

The Offset RMA architecture checks the LPAR effective address against the contents of an RMOL register allowing the access, after adding an LPAR specific offset to form the real address, if the effective address is less, else signaling a protection exception.

14.1.1.2 Reserved Virtual Addresses

The platform may map the RMA through the hashed paged table via a reserved range of virtual addresses. This mapping from the effective address is done by setting the high order virtual address bits corresponding to the VSID to the 0b00 || 0x001FFFFFF 1 TB VSID value. This virtual address is then translated as other virtual addresses. If the effective address is outside the bounds of the RMA, the storage interrupt signals a PTEG miss. The platform firmware prepopulates the LPAR's page frame table with "bolted" entries representing the real storage blocks that make up the RMA. Note, this method allows for the RMA to be discontiguous in real address space. The Virtualized Real Mode Area (VRMA) option gives the OS the ability to dynamically relocate, expand, and shrink the RMA. See Section 14.12.2, "Virtualizing the Real Mode Area," on page 473 for more details.

14.1.2 General LPAR Reservations and Conventions

This section documents general LPAR reserved facilities and conventions. Other sections document reserved facilities and conventions specific to the function they describe.

R1–14.1.2–1. For the LPAR option: To avoid conflict with the platform's reserved addresses, the OS must not use the 1 TB (SLB and PTE B field equal to one) 0b00 || 0x001FFFFFF VSID for purposes other than virtualizing the RMA.

R1–14.1.2–2. For the LPAR option: In order to avoid a storage exception, the OS must not remove PTEs marked with the "bolted" indicator (PTE bit 59 = 1) unless the virtual address space can be referenced by another PTE or the OS does not intend to access the virtual address space.

R1–14.1.2–3. For the LPAR option: To avoid conflict with the platform's hypervisor, the OS must be prepared to share use of SPRG2 as the interrupt scratch register whenever an hcall() is made, or a machine check or reset interrupt is taken.

R1–14.1.2–4. For the LPAR option: If the platform virtualizes the RMA, prior to transferring control to the OS, the platform must select a page size for the RMA such that the platform uses only one page table entry per page table entry group to virtualize the RMA.

R1–14.1.2–5. For the LPAR option: If the platform virtualizes the RMA, prior to transferring control to the OS, the platform must use only the last page table entry of a page table entry group to virtualize the RMA.

14.2 Processor Requirements

R1–14.2–1. For the LPAR option: The platform processors must support the Logical Partitioning (LPAR) facilities as defined in *Power ISA* [1] (Version 2.0 or later).

14.3 I/O Sub-System Requirements

The platform divides the I/O subsystem up into Partitionable Endpoints (PEs). See Section 4.1, "I/O Topologies and Endpoint Partitioning," on page 71 for more information on PEs. Each PE has its own (separate) error, addressing, and interrupt domains which allows the assignment of separate PEs to different LPAR partitions.

The following are the requirements for I/O subsystems when the platform implements LPAR.

R1–14.3–1. For the LPAR option: The platform must provide methods and mechanisms to isolate IOA and I/O bus errors from one PE from affecting another PE, from affecting a partition to which the PE is not given access authority by the platform, and from affecting system resources such as the service processor which are shared between partitions, and must do so with the EEH option programming model.

R1–14.3–2. For the LPAR option: The platform must enable the EEH option for all PEs by default.

Software and Firmware Implementation Notes: For the platform (versus the OS or device driver) to enable EEH, there must be some assurance that the device drivers are EEH aware, if not EEH enabled. For example, the device driver or OS may signal its awareness by using *ibm,set-eeh-option* RTAS call to enable EEH prior to a configuration cycle via the *ibm,write-pci-config* RTAS call which enables the Memory Space or IO Space enable bits in the PCI Command register, and firmware can ignore the *ibm,write-pci-config* RTAS call which enables the Memory Space or IO Space enable bits for an IOA if EEH for that IOA has not been enabled first. To be EEH aware, a device driver does not need to be able to recover from an MMIO Stopped and DMA Stopped state, only recognize the all-1's condition (from a *Load* from its IOA or on a PCI configuration read from its IOA) and not use data from operations that may have occurred since the last all-1's checkpoint. In addition, the device driver under such failure circumstances needs to turn off interrupts (using the *ibm,set-int-off* RTAS call, or for conventional PCI and PCI-X infrastructures only: by resetting the IOA and keeping it reset with *ibm,set-slot-reset* or *ibm,slot-error-detail*) to make sure that any (unserviceable) interrupts from the IOA do not affect the system (MSIs are blocked by the EEH DMA Stopped State, but LSIs are not). Note that if all-1's data may be valid, the *ibm,read-slot-reset-state2* RTAS call should be used to discover the true EEH state of the device.

R1–14.3–3. For the LPAR option: The platform must assign a PE to one and only one partition at a time.

R1–14.3–4. For the LPAR option: The platform must limit the DMA addresses accessible to a PE to the address ranges assigned to the partition to which the PE is allocated, and, if the PE is used to implement a VIO device, then also to any allowed redirected DMA address ranges.

Architecture and Implementation Notes:

1. Platforms which do not implement either Requirement R1–14.3–1 or Requirement R1–14.3–4 require PE granularity of everything below the PHB, resulting in poor LPAR partition I/O assignment granularity.

2. Requirement R1–14.3–4 has implications in preventing access to both to I/O address ranges and system memory address ranges. That is, Requirement R1–14.3–4 requires prevention of peer to peer operations from one IOA to another IOA when those IOA addresses are not owned by the same partition, as well as to providing an access protection mechanism to protect system memory. Note that relative to peer to peer operations, some bridges or switches may not provide the capabilities to limit peer to peer, and the use of such bridges or switches require the limitation that all IOAs under such bridges or switches be assigned to the same partition.

R1–14.3–5. For the LPAR option: The platform must provide a PE the capability of accessing all of the System Memory addresses assigned to the partition to which the PE is allocated.

R1–14.3–6. For the LPAR option: If TCEs are used to satisfy Requirements R1–14.3–4, and R1–14.3–5, then the platform must provide the capability to map simultaneously and at all times at least 256 MB for each PE.

R1–14.3–7. For the LPAR option: If TCEs are used to satisfy Requirements R1–14.3–4, and R1–14.3–5, then the platform must prevent any DMA operations to System Memory addresses which are not translated by TCEs.

R1–14.3–8. For the LPAR option: The DMA address range accessible to a PCI IOA on its I/O bus must be defined by the **`"ibm,dma-window"`** property in its parent's OF device tree node.

Platform Implementation Note: To maximize the ability to migrate memory pages underneath active DMA operations, when ever possible, a bridge should create a bus for a single IOA and either its representing bridge node should include the **`"ibm,dma-window"`** property specific for the IOA for conventional PCI or PCI-X IOAs or the IOA function nodes should contain the **`"ibm,my-dma-window"`** property specific for the IOA function for PCI Express IOAs. When the configuration of a bus precludes memory migration, the platform may combine the DMA address for multiple IOAs that share a bus into a single **`"ibm,dma-window"`** property housed in the bridge node representing the bridge that creates the shared bus.

14.4 Interrupt Sub-System Requirements

R1–14.4–1. For the LPAR option: The platform must not assign the same interrupt (LSI or MSI) or same interrupt source number to different PEs (interrupts cannot be shared between partitions).

R1–14.4–2. For the LPAR option: The interrupt presentation layer must support at least one global interrupt queue per platform supported partition.

R1–14.4–3. For the LPAR option: The interrupt presentation layer must separate the per processor interrupt management areas into a separate 4 K pages per processor so that they can each be individually protected by the PTE mechanism and assigned to their respective assigned partitions.

R1–14.4–4. For the LPAR option: The platform must restrict the processors that service a global queue to those assigned to a single partition.

R1–14.4–5. For the LPAR option: If the interrupt source layer supports message signaled interrupts, the platform must isolate the PCI Message interrupt Input Port (PMIP) in its own 4 K page of the platform's address space.

R1–14.4–6. For the LPAR option: If the interrupt source layer supports message signaled interrupts, the hardware must ignore all writes to the PMIP's 4 K page except those to the PMIP itself.

R1–14.4–7. For the LPAR option: If the interrupt source layer supports message signaled interrupts, the hardware must return all ones on reads of the PMIP's 4 K page except those to the PMIP itself. Signalling a machine check interrupt to the affected processor on a read that returns all 1s as above is optional.

R1–14.4–8. For the LPAR option: The interrupt source layer must support a means in addition to the inter-processor interrupt mechanism for the hypervisor to signal an interrupt to any processor assigned to a partition.

Software Note: While firmware takes all reasonable steps to prevent it, it may be possible, on some hardware implementations, for an OS to erroneously direct an individual IOA's interrupt to another partition's processor. An OS supporting LPAR should ignore such "Phantom" interrupts.

14.5 Hypervisor Requirements

The purpose of the hypervisor is to virtualize the facilities of platforms, with LPAR, such that multiple copies of a LoPAPR compliant OS may simultaneously run protected from each other in different logical partitions of the platform. That is, the various OS images may, without explicit knowledge of each other, boot, run applications, handle exceptions and events and terminate without affecting each other.

The hypervisor is entered by way of three interrupts: the System Reset Interrupt, the Machine Check Interrupt and System (hypervisor) Call Interrupt. These use hypervisor interrupt vectors 0x0100, 0x0200, and 0x0C00 respectively. In addition, a processor implementation dependent interrupt, at its assigned address may cause the hypervisor to be entered. The return from the hypervisor to the OS is via the *rfid* (Return from Interrupt Doubleword) instruction. The target of the *rfid* (instruction at the address contained in SRR0) is either a firmware glue routine (in the case of System Reset or Machine Check) or the instruction immediately following the invoking hypervisor call. The reason for the firmware glue routines is that the OS must do its own processing because of the asynchronous nature of System Reset or Machine Check interruptions. The firmware glue routine calls an OS registered recovery routine for the System Reset or Machine Check condition for further details see (reference to recoverable machine check ACR material to be added when available). The glue routines are registered by the partition's OS through RTAS. Until the glue routines are registered, the OS does not receive direct reports of either System Reset or Machine Check interrupts but is simply re-booted by the hypervisor. The glue routines contain a register buffer area that the hypervisor fills with register values that the glue routine must pass to the OS when calling the interrupt handler. The last element in this buffer is a lock word. The lock word is set with the value of the using processor, and reset by the glue routine just before calling the OS interrupt handler. This way only one buffer is needed per partition rather than one per processor.

At the invocation of the hypervisor, footprint records are generated for recovery conditions. Machine Check and Check Stop conditions are, in some cases, isolatable to the affected partition(s). In these cases, the hypervisor can then prove that it was not executing changes to the global system tables on the offending processor when the error occurred. If this cannot be proven, the global state of the complex is in doubt and the error cannot be contained. It is anticipated that check stops that only corrupt the internal state of the affected processor, stop that processor only. When the service processor subsequently notices the stopped processor it notifies one of the other processors in the partition through a simulated recoverable machine check. The hypervisor running on the notified processor then takes appropriate action to log out and restart the partition, or if there is an alternate cpu capability, then continue execution with a substitute for the stopped processor.

The following table presents the functions supplied by the hypervisor.

Architecture Note: Some functions performed by partition firmware (OF and RTAS) require hypervisor assist, but those firmware implementation dependent interfaces do not appear in this document.

Table 175. Architected hcall()s

Function Name/Section	Comments
H_REMOVE / 14.5.4.1.1	Removes a PTE from the partition's node Page Frame Table
H_BULK_REMOVE / 14.5.4.1.7	Removes up to four (4) PTEs from the partition's node Page Frame Table
H_ENTER / 14.5.4.1.2	Inserts a PTE into the partition's node Page Frame Table

Table 175. Architected hcall()s ***(Continued)***

Function Name/Section	Comments
H_READ / 14.5.4.1.3	Reads the specified PTE from the partition's node Page Frame Table
H_CLEAR_MOD / 14.5.4.1.4	Clears the Modified bit in the specified PTE in the partition's node Page Frame Table
H_CLEAR_REF / 14.5.4.1.5	Clears the Referenced bit in the specified PTE in the partition's node Page Frame Table
H_PROTECT / 14.5.4.1.6	Sets the Page Protection and Storage Key bits in the specified PTE in the partition's node Page Frame Table
H_GET_TCE / 14.5.4.2.1	Returns the value of the specified DMA Translation Control Entry
H_PUT_TCE / 14.5.4.2.2	Inserts the specified value into the specified DMA Translation Control Entry
H_STUFF_TCE / 14.5.4.2.3	Inserts the specified value into multiple DMA Translation Control Entries
H_PUT_TCE_INDIRECT / 14.5.4.2.4	Inserts a list of values into the specified range of DMA Translation Control Entries
H_SET_SPRG0 / 14.5.4.3.1	SPRG0 is architecturally a hypervisor resource. This call allows the OS to write the register.
H_SET_DABR / 14.5.4.3.2	DABR is architecturally a hypervisor resource. This call allows the OS to write the register.
H_PAGE_INIT / 14.5.4.3.3	Initializes pages in real mode either to zero or to the copied contents of another page.
H_SET_XDABR / 14.5.4.3.4	Manage the Extended DABR facility.
H_LOGICAL_CI_LOAD / 14.5.4.4.1	Returns the value contained in a cache inhibited logical address
H_LOGICAL_CI_STORE / 14.5.4.4.2	Stores a value into a cache inhibited logical address
H_GET_TERM_CHAR / 16.6.2.1.1	Returns up to 16 bytes of virtualized console terminal data.
H_PUT_TERM_CHAR / 16.6.2.1.2	Sends up to 16 bytes of data to a virtualized console terminal.
H_VTERM_PARTNER_INFO / 16.6.2.4.2.1	Gets a list of possible client Vterm IOA connections.
H_REGISTER_VTERM / 16.6.2.4.2.2	Associates server Vterm IOA to client Vterm IOA.
H_FREE_VTERM / 16.6.2.4.2.3	Breaks association between server Vterm IOA and client Vterm IOA.
H_HYPERVISOR_DATA / 14.5.4.6.1	Returns internal hypervisor work areas for code maintenance.
H_EOI / 14.5.4.7.1	Generates and End Of Interrupt
H_CPPR / 14.5.4.7.2	Sets the Processor's Current Interrupt Priority
H_IPI / 14.5.4.7.3	Generates an Inter-processor Interrupt
H_IPOLL / 14.5.4.7.4	Polls for pending interrupt
H_XIRR / H_XIRR-X / 14.5.4.7.5	Accepts pending interrupt
H_MIGRATE_DMA / 14.5.4.8.1	Migrates the page underneath an active DMA operation.
H_PERFMON / 14.5.4.9.1	Manages the performance monitor facility.
H_REGISTER_VPA / 14.11.3.2	Registers the virtual processor area that contains the virtual processor dispatch count
H_CEDE / 14.11.3.3	Makes processor virtual processor cycles available for other uses (called when an OS image is idle)
H_CONFER / 14.11.3.4	Causes a virtual processor's cycles to be transferred to a specified processor. (Called by a blocked OS image to allow a lock holder to use virtual processor cycles rather than waiting for the block to clear.)
H_PROD / 14.11.3.5	Awakens a virtual processor that has ceded its cycles.

Table 175. Architected hcall()s ***(Continued)***

Function Name/Section	Comments
H_GET_PPP / 14.11.3.6	Returns the partition's virtual processor performance parameters.
H_SET_PPP / 14.11.3.7	Sets the partition's virtual processor performance parameters (within constraints).
H_PURR / 14.11.3.8	Returns the value of the virtual processor utilization register.
H_POLL_PENDING / 14.11.3.9	Polls the hypervisor for the existence of pending work to dispatch on the calling processor.
H_PIC / 14.11.4.1	Returns the summation of the physical processor pool's idle cycles.
H_REG_CRQ / 17.2.3.1.5.1	Register Command/Response Queue
H_FREE_CRQ / 17.2.3.1.5.2	Frees the memory associated with the Command/Response Queue
H_VIO_SIGNAL / 17.2.1.3.1	Controls the virtual interrupt signaling of virtual IOAs
H_SEND_CRQ / 17.2.3.1.5.3	Sends a message on the Command/Response Queue
H_PUT_RTCE / 17.2.2.2.1	Loads a Redirected Remote DMA Remote Translation Control Entry
H_PUT_RTCE_INDIRECT / 17.2.2.2.2	Loads a list of Redirected Remote DMA Remote Translation Control Entries
H_REMOVE_RTCE / 17.2.2.2.3	Unmaps a redirected TCE that was previously built with H_PUT_RTCE or H_PUT_RTCE_INDIRECT
H_LIOBN_ATTRIBUTES / 17.2.2.2.6	Allows modification of LIOBN Attributes.
H_COPY_RDMA / 17.2.3.2.1.1	Copies data between partitions as if by TCE mapped DMA.
H_WRITE_RDMA / 17.2.3.2.1.2	Write parameter data to remote DMA buffer.
H_READ_RDMA / 17.2.3.2.1.3	Read data from remote DMA buffer to return registers.
H_REGISTER_LOGICAL_LAN / 16.4.3.1	Registers the partition's logical LAN control structures with the hypervisor
H_FREE_LOGICAL_LAN / 16.4.3.2	Releases the partition's logical LAN control structures
H_ADD_LOGICAL_LAN_BUFFER / 16.4.3.3	Adds receive buffers to the logical LAN receive buffer pool
H_SEND_LOGICAL_LAN / 16.4.3.5	Sends a logical LAN message
H_MULTICAST_CTRL / 16.4.3.6	Controls the reception and filtering of non-broadcast multicast packets.
H_CHANGE_LOGICAL_LAN_MAC / 16.4.3.7	Changes the MAC address for an ILLAN virtual IOA.
H_ILLAN_ATTRIBUTES / 16.4.3.8	Allows modifications of ILLAN Attributes.
/	Map from 4 KB up to the size of a full LMB per RTCE table TCE entry, and map multiple TCEs (and therefore multiple LMBs) in one operation.
H_ALRDMA / 17.2.4.2.9	ALRDMA CRQ and ASQ setup and control.
H_GRANT_LOGICAL / 17.2.1.5.1	Constructs a cookie, specific to the intended client, representing a shared resource.
H_RESCIND_LOGICAL / 17.2.1.5.2	Invalidates a cookie representing a shared resource.
H_ACCEPT_LOGICAL / 17.2.1.5.3	Maps a shared resource into the client's logical address space
H_RETURN_LOGICAL / 17.2.1.5.4	Removes a shared resource from a client's logical address space.
H_FREE_LOGICAL_LAN_BUFFER / 16.4.3.4	Removes receive buffers of specified size from the logical LAN receive buffer pool.
H_VIOCTL / 17.2.1.6	Allows the partition to manipulate or query certain virtual IOA behaviors.

Table 175. Architected hcall()s *(Continued)*

Function Name/Section	Comments
H_JOIN / 14.11.5.1	Join active threads and return H_CONTINUE to final calling thread
H_DONOR_OPERATION / 17.8.6.1	Use the calling processor to perform platform operations.
H_VASI_SIGNAL / 17.8.6.2	Transition VASI operation stream state.
H_VASI_STATE / 17.8.6.3	Return the VASI operation stream state.
H_ENABLE_CRQ / 17.2.3.1.5.4	Reactivate a suspended CRQ.
H_VRMASD / 14.12.2.1	Change the page mapping characteristics of the Virtualized Real Mode Area.
H_VPM_PSTAT / 14.12.4.1	Returns Virtual Partition Memory pool statistics
H_SET_MPP / 14.12.3.4	Set Memory Performance Parameters
H_GET_MPP / 14.12.3.5	Get Memory Performance Parameters
H_MO_PERF / 14.12.3.7	Determine Memory Overcommit Performance
H_REG_SUB_CRQ / 17.2.3.3.5.1	Register a Sub-CRQ.
H_FREE_SUB_CRQ / 17.2.3.3.5.2	Free a Sub-CRQ.
H_SEND_SUB_CRQ / 17.2.3.3.5.3	Send a message to a Sub-CRQ.
H_SEND_SUB_CRQ_INDIRECT / 17.2.3.3.5.4	Send a list of messages to a Sub-CRQ.
H_HOME_NODE_ASSOCIATIVITY / 14.11.6.1	Report the home node associativity for a given virtual processor
H_GET_EM_PARMS / 14.14.2	Get the partition energy management parameters
H_BEST_ENERGY / 14.14.2.1	Returns hints for activating/releasing resource instances to achieve the best energy efficiency.
SNS Registration (H_REG_SNS) / 14.12.3.8.3.2	Registers Subvention Notification Structure
H_RANDOM / 14.15.1	Get a random number
14.15.2.1 H_COP_OP: / 14.15.2.1	Initiate a co-processor operation
14.15.2.2 H_STOP_COP_OP / 14.15.2.2	Stop a co-processor operation
H_GET_MPP_X / 14.12.3.5.1	Get Extended Memory Performance Parameters
H_SET_MODE / 14.5.4.3.5	Set Processing resource mode
H_GET_DMA_XLATES_LIMITED / 14.5.4.10	Search TCE table for entries within a specified range

14.5.1 System Reset Interrupt

Hypervisor code saves all processor state by saving the contents of one register in SPRG2 (SPRG1 if *ibm,nmi-register-2* was used) (Multiplexing the use of this resource with the OS). The processor's stack and data area are found by processing the Processor Identification Register.

R1–14.5.1–1. For the LPAR option: The platform must support signalling system reset interrupts to all processors assigned to a partition.

R1–14.5.1–2. For the LPAR option: The platform must support signalling system reset interrupts individually as well as collectively to all supported partitions.

R1–14.5.1–3. For the LPAR option: The system reset interrupts signaled to one partition must not affect operations of another partition.

R1–14.5.1–4. For the LPAR option: The hypervisor must intercept all system reset interrupts.

R1–14.5.1–5. For the LPAR option: The platform must implement the FWNMI option.

R1–14.5.1–6. For the LPAR option: The hypervisor must maintain a count, reset when the partition's OS, through RTAS, registers for system reset interrupt notification, of system reset interrupts signaled to a partition's processor.

R1–14.5.1–7. For the LPAR option: Once the partition's OS has registered for system reset interrupt notification, the hypervisor must forward the first and second system reset interrupts signaled to a partition's processor.

R1–14.5.1–8. For the LPAR option: The hypervisor must on the third and all subsequent system reset interrupts signaled to a partition's processor invoke OF to initiate the partition's reboot policy.

R1–14.5.1–9. For the LPAR option: The hypervisor must have the capability to receive and handle the system reset interrupts simultaneously on multiple processors in the same or different partitions up to the number of processors in the system.

14.5.2 Machine Check Interrupt

Hypervisor code saves all processor state by saving the contents of one register in SPRG2 (SPRG1 if *ibm,nmi-register-2* was used) (Multiplexing the use of this resource with the OS). The processor's stack and data area are found by processing the Processor Identification Register.

The hypervisor investigates the cause of the machine check. The cause is either a recoverable event on the current processor, or a non-recoverable event either on the current processor or one of the other processors in the logical partition. Also the hypervisor must determine if the machine check may have corrupted its own internal state (by looking at the footprints, if any, that were left in the per processor data area of the errant processor.

R1–14.5.2–1. For the LPAR option: The hypervisor must have the capability to receive and handle the machine check interrupts simultaneously on multiple processors in the same or different partitions up to the number of processors in the system.

14.5.3 Hypervisor Call Interrupt

The hypervisor call (hcall) interrupt is a special variety of the system call instruction. The parameters to the hcall() are passed in registers using the PA ABI definitions (Reg 3-12 for parameters). In contrast to the PA ABI, pass by reference parameters are avoided to or from hcall(). This minimizes the address translation problem pointer parameters cause. Some input parameters are indexes. Output parameters, when generated, are passed in registers 4 through 12 and require special in-line assembler code on the part of the caller. The first parameter to hcall() is the function token. Table 176, "Hypervisor Call Function Table," on page 394 specifies the valid hcall() function names and token values. Some of the hcall() functions are optional, to indicate if the platform is in LPAR mode, and which functions are available on a given platform, the OF property **`"ibm,hypertas-functions"`** is provided in the **`/rtas`** node of the partition's device tree. The property is present if the platform is in LPAR mode while its value specifies which function sets are implemented by a given implementation. If platform implements any hcall() of a function set it implements the entire function set. Additionally, certain values of the **`"ibm,hypertas-functions"`** property indicate that the platform supports a given architecture extension to a standard hcall().

The floating point registers along with the FPSCR are in general preserved across hcall() functions, unless the "Maintain FPRs" field of the VPA =0, see Table 184, "Per Virtual Processor Area," on page 449. The general purpose registers r0 and r3-r12, the CTR and XER registers are volatile along with the condition register fields 0 and 1 plus 5-7. Specific hcall()s may specify a more restricted "kill set", refer to the specific hcall() specification below.

R1–14.5.3–1. For the LPAR option: The platform's **/rtas** node must contain an **"ibm,hypertas-functions"** property as defined below.

R1–14.5.3–2. For the LPAR option: If a platform reports in its **"ibm,hypertas-functions"** property (see Section B.6.3.1, "RTAS Node Properties," on page 690) that it supports a function set, then it must support all hcall()s of that function set as defined in Table 176, "Hypervisor Call Function Table," on page 394.

Table 176. Hypervisor Call Function Table

Hypervisor Call Function Name/Section	Hypervisor Call Function Token	Hypervisor Call Performance Class	Function Mandatory?	Function Set
UNUSED	0x0			
H_REMOVE / 14.5.4.1.1	0x4	Critical	Yes	hcall-pft
H_ENTER / 14.5.4.1.2	0x8	Critical	Yes	hcall-pft
H_READ / 14.5.4.1.3	0xC	Critical	Yes	hcall-pft
H_CLEAR_MOD / 14.5.4.1.4	0x10	Critical	Yes	hcall-pft
H_CLEAR_REF / 14.5.4.1.5	0x14	Critical	Yes	hcall-pft
H_PROTECT / 14.5.4.1.6	0x18	Critical	Yes	hcall-pft
H_GET_TCE / 14.5.4.2.1	0x1C	Critical	Yes	hcall-tce
H_PUT_TCE / 14.5.4.2.2	0x20	Critical	Yes	hcall-tce
H_SET_SPRG0 / 14.5.4.3.1	0x24	Critical	Yes	hcall-sprg0
H_SET_DABR / 14.5.4.3.2	0x28	Critical	Yes - if DABR exists	hcall-dabr
H_PAGE_INIT / 14.5.4.3.3	0x2C	Critical	Yes	hcall-copy
H_LOGICAL_CI_LOAD / 14.5.4.4.1	0x3C	Normal	Yes	hcall-debug
H_LOGICAL_CI_STORE / 14.5.4.4.2	0x40	Normal	Yes	hcall-debug
H_GET_TERM_CHAR / 16.6.2.1.1	0x54	Critical	Yes	hcall-term
H_PUT_TERM_CHAR / 16.6.2.1.2	0x58	Critical	Yes	hcall-term
H_HYPERVISOR_DATA / 14.5.4.6.1	0x60	Normal	Yes if enabled by HMC (default disabled)	hcall-dump
H_EOI / 14.5.4.7.1	0x64	Critical	Yes	hcall-interrupt
H_CPPR / 14.5.4.7.2	0x68	Critical	Yes	hcall-interrupt
H_IPI / 14.5.4.7.3	0x6C	Critical	Yes	hcall-interrupt
H_IPOLL / 14.5.4.7.4	0x70	Critical	Yes	hcall-interrupt
H_XIRR / 14.5.4.7.5	0x74	Critical	Yes	hcall-interrupt
H_XIRR-X 14.5.4.7.5	0x2FC	Critical	Yes	hcall-interrupt
H_MIGRATE_DMA / 14.5.4.8.1	0x78	Normal	If LRDR option is implemented	hcall-migrate
H_PERFMON / 14.5.4.9.1	0x7C	Normal	If performance monitor is implemented	hcall-perfmon

Table 176. Hypervisor Call Function Table ***(Continued)***

Hypervisor Call Function Name/Section	Hypervisor Call Function Token	Hypervisor Call Performance Class	Function Mandatory?	Function Set
Reserved	0x80 - 0xD8			
H_REGISTER_VPA / 14.11.3.2	0xDC	Normal	If SPLPAR or SLB Shadow Buffer option is implemented	hcall-splpar SLB-Buffer
H_CEDE / 14.11.3.3	0xE0	Critical	If SPLPAR option is implemented	hcall-splpar
H_CONFER / 14.11.3.4	0xE4	Critical	If SPLPAR option is implemented	hcall-splpar
H_PROD / 14.11.3.5	0xE8	Critical	If SPLPAR option is implemented	hcall-splpar
H_GET_PPP / 14.11.3.6	0xEC	Normal	If SPLPAR option is implemented	hcall-splpar
H_SET_PPP / 14.11.3.7	0xF0	Normal	If SPLPAR option is implemented	hcall-splpar
H_PURR / 14.11.3.8	0xF4	Critical	If SPLPAR option is implemented	hcall-splpar
H_PIC / 14.11.4.1	0xF8	Normal	If SPLPAR option is implemented	hcall-pic
H_REG_CRQ / 17.2.3.1.5.1	0xFC	Normal	If VSCSI option is implemented	hcall-crq
H_FREE_CRQ / 17.2.3.1.5.2	0x100	Normal	If VSCSI option is implemented	hcall-crq
H_VIO_SIGNAL / 17.2.1.3.1	0x104	Critical	If either the VSCSI or logical LAN option is implemented	hcall-vio
H_SEND_CRQ / 17.2.3.1.5.3	0x108	Critical	If VSCSI option is implemented	hcall-crq
H_PUT_RTCE / 17.2.2.2.1	0x10C	Critical	If VSCSI option is implemented	hcall-rdma
H_COPY_RDMA / 17.2.3.2.1.1	0x110	Critical	If VSCSI option is implemented	hcall-rdma
H_REGISTER_LOGICAL_LAN / 16.4.3.1	0x114	Normal	If logical LAN option is implemented	hcall-lLAN
H_FREE_LOGICAL_LAN / 16.4.3.2	0x118	Normal	If logical LAN option is implemented	hcall-lLAN
H_ADD_LOGICAL_LAN_BUFFER / 16.4.3.3	0x11C	Critical	If logical LAN option is implemented	hcall-lLAN
H_SEND_LOGICAL_LAN / 16.4.3.5	0x120	Critical	If logical LAN option is implemented	hcall-lLAN
H_BULK_REMOVE / 14.5.4.1.7	0x124	Critical	New designs as of 01/01/2003	hcall-bulk
H_WRITE_RDMA / 17.2.3.2.1.2	0x128	Critical	If VSCSI option is implemented	hcall-rdma

Table 176. Hypervisor Call Function Table *(Continued)*

Hypervisor Call Function Name/Section	Hypervisor Call Function Token	Hypervisor Call Performance Class	Function Mandatory?	Function Set
H_READ_RDMA / 17.2.3.2.1.3	0x12C	Critical	If VSCSI option is implemented	hcall-rdma
H_MULTICAST_CTRL / 16.4.3.6	0x130	Critical	If logical LAN option is implemented	hcall-lLAN
H_SET_XDABR / 14.5.4.3.4	0x134	Normal	If Extended DABR option is implemented	hcall-xdabr
H_STUFF_TCE / 14.5.4.2.3	0x138	Critical		hcall-multi-tce
H_PUT_TCE_INDIRECT / 14.5.4.2.4	0x13C	Critical		hcall-multi-tce
H_PUT_RTCE_INDIRECT / 17.2.2.2.2	0x140	Critical		hcall-multi-tce
Reserved	0x144			
Reserved	0x148			
H_CHANGE_LOGICAL_LAN_MAC / 16.4.3.7	0x14C	Normal	If Logical LAN option is implemented	hcall-ILAN
H_VTERM_PARTNER_INFO / 16.6.2.4.2.1	0x150	Normal	If the Server Vterm option is implemented	hcall-vty
H_REGISTER_VTERM / 16.6.2.4.2.2	0x154	Normal	If the Server Vterm option is implemented	hcall-vty
H_FREE_VTERM / 16.6.2.4.2.3	0x158	Normal	If the Server Vterm option is implemented	hcall-vty
H_GRANT_LOGICAL / 17.2.1.5.1	0x1C4	Normal	If Shared Logical Resource option is Implemented	hcall-slr
H_RESCIND_LOGICAL / 17.2.1.5.2	0x1C8	Normal	If Shared Logical Resource option is Implemented	hcall-slr
H_ACCEPT_LOGICAL / 17.2.1.5.3	0x1CC	Normal	If Shared Logical Resource option is Implemented	hcall-slr
H_RETURN_LOGICAL / 17.2.1.5.4	0x1D0	Normal	If Shared Logical Resource option is Implemented	hcall-slr
H_FREE_LOGICAL_LAN_BUFFER / 16.4.3.4	0x1D4	Critical	If logical LAN option is implemented	hcall-lLAN
H_POLL_PENDING / 14.11.3.9	0x1D8	Critical	If SPLPAR option is implemented	hcall-poll-pending
Reserved	0x1DC - 0x1E0	Varies		
Reserved	0x1E8 - 0x1EC			
Reserved	0x1F0 - 0x23C	Varies		
H_LIOBN_ATTRIBUTES / 17.2.2.2.6	0x240	Normal	If LIOBN Attributes are implemented	hcall-liobn-attributes

Table 176. Hypervisor Call Function Table *(Continued)*

Hypervisor Call Function Name/Section	Hypervisor Call Function Token	Hypervisor Call Performance Class	Function Mandatory?	Function Set
H_ILLAN_ATTRIBUTES / 16.4.3.8	0x244	Normal	If ILLAN Checksum Offload Support is implemented If ILLAN Backup Trunk Adapter option is implemented	hcall-illan-options
Reserved	0x248			
H_REMOVE_RTCE / 17.2.2.2.3	0x24C	Critical	If H_PUT_RTCE is implemented	hcall-rdma
Reserved	0x27C			
Reserved	0x280			
Reserved	0x28C-0x294			
H_JOIN / 14.11.5.1	0x298	Normal	If Thread Join option is implemented	hcall-join
H_DONOR_OPERATION / 17.8.6.1	0x29C	Normal	If VASI option is implemented	hcall-vasi
H_VASI_SIGNAL / 17.8.6.2	0x2A0	Normal	If VASI option is implemented	hcall-vasi
H_VASI_STATE / 17.8.6.3	0x2A4	Normal	If VASI option is implemented	hcall-vasi
H_VIOCTL / 17.2.1.6	0x2A8	Normal	If any virtual I/O options are implemented	hcall-vioctl
H_VRMASD / 14.12.2.1	0x2AC	Normal	If the VRMA option is implemented.	hcall-vrma
H_ENABLE_CRQ / 17.2.3.1.5.4	0x2B0	Continued	If partition suspension option is implemented	hcall-suspend
Reserved	0x2B4			
H_GET_EM_PARMS / 14.14.2	0x2B8	Normal	If the Partition Energy Management Option is implemented	hcall-get-emparm
H_VPM_PSTAT / 14.12.4.1	0x2BC	Normal	If the Cooperative Memory Over-commitment Option is implemented	hcall-cmo
H_SET_MPP / 14.12.3.4	0x2D0	Normal	If the Cooperative Memory Over-commitment Option is implemented	hcall-cmo
H_GET_MPP / 14.12.3.5	0x2D4	Normal	If the Cooperative Memory Over-commitment Option is implemented	hcall-cmo

Table 176. Hypervisor Call Function Table *(Continued)*

Hypervisor Call Function Name/Section	Hypervisor Call Function Token	Hypervisor Call Performance Class	Function Mandatory?	Function Set
H_MO_PERF / 14.12.3.7	0x2D8	Normal	If the Cooperative Memory Over-commitment Option is implemented and the calling partition is authorized.	hcall-cmo
H_REG_SUB_CRQ / 17.2.3.3.5.1	0x2DC	Normal	If the Subordinate CRQ Option is implemented	hcall-sub-crq
H_FREE_SUB_CRQ / 17.2.3.3.5.2	0x2E0	Normal	If the Subordinate CRQ Option is implemented	hcall-sub-crq
H_SEND_SUB_CRQ / 17.2.3.3.5.3	0x2E4	Normal	If the Subordinate CRQ Option is implemented	hcall-sub-crq
H_SEND_SUB_CRQ_INDIRECT / 17.2.3.3.5.4	0x2E8	Normal	If the Subordinate CRQ Option is implemented	hcall-sub-crq
H_HOME_NODE_ASSOCIATIVITY / 14.11.6.1	0x2EC	Normal	If the VPHN Option is implemented	hcall-vphn
Reserved	0x2F0			
H_BEST_ENERGY / 14.14.2.1	0x2F4	Normal	If the Partition Energy Management Option is implemented	hcall-best-energy-1<list> [a]
SNS Registration (H_REG_SNS) / 14.12.3.8.3.2	0x2F8	Normal	If the Expropriation Subvention Notification Option is implemented	hcall-esn
X_XIRR-X / 14.5.4.7.5	0x2FC	Critical	Yes	hcall-interrupt
H_RANDOM / 14.15.1	0x300	Normal	If a random number generator Platform Facilities Option is implemented	hcall-random
Reserved	0x310			
14.15.2.1 H_COP_OP: / 14.15.2.1	0x304	Normal	If one or more Coprocessor Platform Facilities Options are implemented	hcall-cop
14.15.2.2 H_STOP_COP_OP / 14.15.2.2	0x308	Normal	If one or more Coprocessor Platform Facilities Options are implemented	hcall-cop
H_GET_MPP_X / 14.12.3.5.1	0x314	Normal	If the Extended Cooperative Memory Overcommittment Option is implemented	hcall-cmo-x
H_SET_MODE / 14.5.4.3.5	0x31C	Normal	If the platform supports POWER ISA version 2.07 or higher	hcall-set-mode
Reserved	0x320			

Table 176. Hypervisor Call Function Table *(Continued)*

Hypervisor Call Function Name/Section	Hypervisor Call Function Token	Hypervisor Call Performance Class	Function Mandatory?	Function Set
H_GET_DMA_XLATES_LIMITED / 14.5.4.10	0x324	normal	If the plaform implements the LRDR option at LoPAPR Version 2.7 or higher	hcall-xlates-limited
Reserved for platform-dependent hcall()s / Appendix J	0xF000 - 0xFFFC			
ILLEGAL	Any token value having a one in either of the low order two bits			
Reserved	0x328 - 0xEFFC and 0x10000 - 0xFFFFFFFFFFFFFFFC: RTAS implementations may assigns values in these ranges to their own internal interfaces, as long as they are prepared for the growth of architected functions into this range.			

a. The <list> suffix for hcall-best-energy indicates an optional dash delimited series (may be null) of supported resource codes encoded as ASCII decimal values in addition to the minimal support value of 1 for processors, other values are define in Section 14.14.2.1, “H_BEST_ENERGY,” on page 495.

Firmware Implementation Note: The assignment of function tokens is designed such that a single mask operation can validate that the value is within the range of a reasonable size branch table. Entries within the branch table can handle unimplemented code points.

The hypervisor routines are optimized for execution speed. In some rare cases, locks are taken, and specific hardware designs require short wait loops. However, if a needed resource is truly busy, or processing is required by an agent, the hypervisor returns to the caller, either to have the function retried or continued at a later time. The Performance Class establishes specific performance requirements against each specific hcall() function as defined below.

Hypervisor Call Performance Classes:

Critical — Must make continuous forward progress, encountering any busy resource must cause the function to back out and return with a “hardware busy” return code. When subsequently called, the operation begins again. Short loops for larwx and stwcx to acquire an apparently unheld lock are allowed. These functions may not include wait loops for slow hardware access.

Normal — Similar to critical, however, wait loops for slow hardware access are allowed. These functions may not include wait loops for an agent such as an external micro-processor or message transmission device.

Continued — This class of functions is expected to serialize on the use of external agents. If the external agent is busy the function returns “hardware busy”. If the interface to the external agent is not busy, the interface is marked busy and used to start the function. The function returns one of the “function in progress” return codes. Later, the caller may check on the completion of the function by issuing the “check” Hcall function with the “function in progress” parameter code. If

the function completed properly, the hypervisor maintains no status and the "check" Hcall returns success. If the operation is still in process, the same "function in progress" code is returned. If the function completed in error, the completion error code is returned. The hypervisor maintains room for at least one outstanding error status per external agent interface per processor. If there is no room to record the error status, the hypervisor returns "hardware busy" and does not start the function.

Terminal — This class of functions is used to manage a partition when the OS is not in regular operation. These events include postmortems and extensive recoveries.

The hypervisor performance classes are ordered in decreasing restriction.

R1–14.5.3–3. For the LPAR option: The caller must perform properly given that the hypervisor meets the performance class specified.

R1–14.5.3–4. For the LPAR option: The hypervisor implementation must meet the specified performance class or higher.

R1–14.5.3–5. For the LPAR option: Platform hardware designs must take the allowable performance classes into account when choosing the hardware access technology for the various facilities.

R1–14.5.3–6. For the LPAR option: The hypervisor must have the capability to receive and handle the hypervisor call interrupts simultaneously on multiple processors in the same or different partitions up to the number of processors in the system.

R1–14.5.3–7. For the LPAR option: The hypervisor must check the state of the MSR register bits that are not set to a specific value by the processor hardware during the invoking interrupt per Table 177, "MSR State on Entrance to Hypervisor," on page 400.

Table 177. MSR State on Entrance to Hypervisor

MSR Bit	Required State	Error-Code
HV - Hypervisor	1	None
Bits 2,4:46, 57, and 60 Reserved	Set to 0 by Hardware	None
ILE - Interrupt Little Endian	As Last set by the hypervisor	None
ME - Machine check Enable	As last set by the hypervisor	None
LE Little-Endian Mode	0 forced by ILE	None

R1–14.5.3–8. For the LPAR option: The Hcall() flags field must meet the definition in: Table 178, "Page Frame Table Access flags field definition," on page 401; the hypervisor may safely ignore flag field values not explicitly defined by the specific hcall() semantic.

R1–14.5.3–9. For the LPAR option: The platform must ensure that flag field values not defined for a specific hcall() do not compromise partitioning integrity.

R1–14.5.3–10. For the LPAR option: Implementations that normally choose to ignore invalid flag field values must provide a "debug mode" that does check for invalid flag field values and returns H_Parameter when any are found.

Architecture Note: The method for invocation of a platform's "debug mode" is beyond the scope of this architecture.

Table 178. Page Frame Table Access flags field definition

Bit	Function	Bit	Function	Bit	Function	Bit	Function
0-15	NUMA CEC Cookie	16-23	Subfunction Codes	32	AVPN	48	Zero Page
				33	andcond	49	Copy Page
				34-39	Reserved	50-54	key0-key4[a]
						55	pp0[b]
		24	Exact	40	I-Cache-Invalidate	56	Compression
		25	R-XLATE	41	I-Cache-Synchronize	57	Checksum
		26	READ-4	42	CC (Coalesce Candidate)	58-60	Reserved
		27	Reserved	43-47	Reserved		
		28-31	CMO Option Flags			61	N
						62	pp1
						63	pp2

a. Bits 50-54 (key0 - key4) shall be treated as reserved on platforms that either do not contain an **"ibm,processor-storage-keys"** property, or contain an **"ibm,processor-storage-keys"** property with the value of zero in both cells.
b. Bit 55 (pp0) shall be treated as reserved on platforms that do not have the "Support for the "110" value of the Page Protection (PP) bits" bit set to a value of 1 in the **"ibm,pa-features"** property.

R1–14.5.3–11. For the LPAR option: The caller of Hcall must be in privileged mode (MSR_{PR} = 0) or the hypervisor immediately returns an H_Privilege return code. See Table 179, "Hypervisor Call Return Code Table," on page 402 for this and other architected return codes.

R1–14.5.3–12. For the LPAR option: The caller of hcall() must be prepared for a return code of H_Hardware from all functions.

R1–14.5.3–13. For the LPAR option: In order for the platform to return H_Hardware, the error must not have resulted in an undetectable state/data corruption nor will continued operation propagate an undetectable state/data corruption as a result of the original error.

Notes:

1. A detectable corruption, when accessed, results in either a H_Hardware return code, machine check or check stop per platform policy.

2. Among other implications of Requirement R1–14.5.3–13 are: the effective state of the partition appears to not change due to the failed hcall() -- (any partial changes to persistent state/data are backed out); and the recovery of platform resources that held lost state/data does not hide the state/data loss to subsequent users of that state/data.

3. The operating system is not expected to log a serviceable event due to an H_Hardware return code from an hcall(), and treats the hcall() as failing due to nonspecific hardware reasons. Any logging of a serviceable event in response to the underlying cause is handled by separate platform initiated operations.

Table 179. Hypervisor Call Return Code Table

Hypervisor Call Return Code Values (R3)	Meaning
0x0100000 - 0x0FFFFFF	Function in Progress
9905	H_LongBusyOrder100sec - Similar to LongBusyOrder1msec, but the hint is 100 second wait this time.
9904	H_LongBusyOrder10sec - Similar to LongBusyOrder1msec, but the hint is 10 second wait this time.
9903	H_LongBusyOrder1Sec - Similar to LongBusyOrder1msec, but the hint is 1 second wait this time.
9902	H_LongBusyOrder100mSec - Similar to LongBusyOrder1msec, but the hint is 100mSec wait this time.
9901	H_LongBusyOrder10mSec - Similar to LongBusyOrder1msec, but the hint is 10mSec wait this time.
9900	H_LongBusyOrder1msec - This return code is identical to H_Busy, but with the added bonus of a hint to the partition OS. If the partition OS can delay for 1 millisecond, the hcall will likely succeed on a new hcall with no further busy return codes. If the partition OS cannot handle a delay, they are certainly free to immediately turn around and try again.
18	H_CONTINUE
17	H_PENDING
16	H_PARTIAL_STORE
15	H_PAGE_REGISTERED
14	H_IN_PROGRESS
13	Sensor value >= Critical high
12	Sensor value >= Warning high
11	Sensor value normal
10	Sensor value <= Warning low
9	Sensor value <= Critical low
5	H_PARTIAL (The request completed only partially successful. Parameters were valid but some specific hcall function condition prevented fully completing the architected function, see the specific hcall definition for possible reasons.)
4	H_Constrained (The request called for resources in excess of the maximum allowed. The resultant allocation was constrained to maximum allowed)
3	H_NOT_AVAILABLE
2	H_Closed (virtual I/O connection is closed)
1	H_Busy Hardware Busy -- Retry Later
0	H_Success
-1	H_Hardware (Error)

Table 179. Hypervisor Call Return Code Table ***(Continued)***

Hypervisor Call Return Code Values (R3)	Meaning
-2	H_Function (Not Supported)
-3	H_Privilege (Caller not in privileged mode).
-4	H_Parameter (Outside Valid Range for Partition or conflicting)
-5	bad_mode (Illegal MSR value)
-6	H_PTEG_FULL (The requested pteg was full)
-7	H_Not_Found (The requested entity was not found)
-8	H_RESERVED_DABR (The requested address is reserved by the hypervisor on this processor)
-9	H_NOMEM
-10	H_AUTHORITY (The caller did not have authority to perform the function)
-11	H_Permission (The mapping specified by the request does not allow for the requested transfer)
-12	H_Dropped (One or more packets could not be delivered to their requested destinations)
-13	H_S_Parm (The source parameter is illegal)
-14	H_D_Parm (The destination parameter is illegal)
-15	H_R_Parm (The remote TCE mapping is illegal)
-16	H_Resource (One or more required resources are in use)
-17	H_ADAPTER_PARM (invalid adapter)
-18	H_RH_PARM (resource not valid or logical partition conflicting)
-19	H_RCQ_PARM (RCQ not valid or logical partition conflicting)
-20	H_SCQ_PARM (SCQ not valid or logical partition conflicting)
-21	H_EQ_PARM (EQ not valid or logical partition conflicting)
-22	H_RT_PARM (invalid resource type)
-23	H_ST_PARM (invalid service type)
-24	H_SIGT_PARM (invalid signalling type)
-25	H_TOKEN_PARM (invalid token)
-27	H_MLENGTH_PARM (invalid memory length)
-28	H_MEM_PARM (invalid memory I/O virtual address)
-29	H_MEM_ACCESS_PARM (invalid memory access control)
-30	H_ATTR_PARM (invalid attribute value)
-31	H_PORT_PARM (invalid port number)
-32	H_MCG_PARM (invalid multicast group)

Table 179. Hypervisor Call Return Code Table *(Continued)*

Hypervisor Call Return Code Values (R3)	Meaning
-33	H_VL_PARM (invalid virtual lane)
-34	H_TSIZE_PARM (invalid trace size)
-35	H_TRACE_PARM (invalid trace buffer)
-36	H_TRACE_PARM (invalid trace buffer)
-37	H_MASK_PARM (invalid mask value)
-38	H_MCG_FULL (multicast attachments exceeded)
-39	H_ALIAS_EXIST (alias QP already defined)
-40	H_P_COUNTER (invalid counter specification)
-41	H_TABLE_FULL (resource page table full)
-42	H_ALT_TABLE (alternate table already exists / alternate page table not available)
-43	H_MR_CONDITION (invalid memory region condition)
-44	H_NOT_ENOUGH_RESOURCES (insufficient resources)
-45	H_R_STATE (invalid resource state condition or sequencing error)
-46	H_RESCINDED
-54	H_Aborted
-55	H_P2
-56	H_P3
-57	H_P4
-58	H_P5
-59	H_P6
-60	H_P7
-61	H_P8
-62	H_P9
-63	H_NOOP
-64	H_TOO_BIG
-65	Reserved
-66	Reserved
-67	H_UNSUPPORTED (Parameter value outside of the range supported by this implementation)
-68	H_OVERLAP (unsupported overlap among passed buffer areas)
-69	H_INTERRUPT (Interrupt specification is invalid)

Table 179. Hypervisor Call Return Code Table *(Continued)*

Hypervisor Call Return Code Values (R3)	Meaning
-70	H_BAD_DATA (uncorrectable data error)
-71	H_NOT_ACTIVE (Not associated with an active operation)
-72	H_SG_LIST (A scatter/gather list element is invalid)
-73	H_OP_MODE (There is a conflict between the subcommand and the requested operation notification)
-74	H_COP_HW (co-processor hardware error)
-75	H_STATE (invalid state)
-76	H_RESERVED (a reserved value was specified)
-77 : -255	Reserved
-256 -- -511	H_UNSUPPORTED_FLAG (An unsupported binary flag bit was specified. The returned value is -256 - the bit position of the unsupported flag bit [high order flag bit is 0 etc.])

14.5.4 Hypervisor Call Functions

14.5.4.1 Page Frame Table Access

All hypervisor Page Frame Table (PFT) access routines are called using 64 bit linkage conventions and apply to all page sizes that the platform supports as specified by the **"ibm,processor-page-sizes"** property (See Appendix C, "PA Processor Binding," on page 753 for more details). The hypervisor PFT access functions carefully update a given Page Table Entry (PTE) with at least 64 bit store operations since an invalid update sequence could result in machine checks. To guard against multiple conflicting allocations of a PTE that could result in a check stop condition, the hypervisor PTE allocation routine (H_ENTER) reserves the first two (high order) software PTE bits for use as PTE locks while the low order two software PTE bits are reserved for OS use (not used by firmware). If a firmware PTE bit is on, the OS is to assume that the PTE is in use, just as if the V bit were on. The hypervisor PFT access routines often execute the tlbie instruction, on certain platforms, this instruction may only be executed by one processor in a partition at a time, the hypervisor uses locks to assure this. The tlbie instruction flushes a specific translate lookaside buffer (TLB) entry from all processors participating in the protocol. All the processors participating in the tlbie protocol are defined as a translation domain. All processors of a given partition that are in a given translation domain share the same hardware PFT. Book III of the PA specifies the codes sequences needed to safely access the PFT, in its chapter titled "Storage Control Instructions and Table Updates". These code sequences are part of this specification by reference. The hypervisor PFT access routines are in the critical performance path of the machine, therefore, extraordinary care must be given to their performance, including machine dependent coding, minimal run time checking, and code path length optimization. For performance reasons, all parameter linkage is through registers, and no indirect parameter linkage is allowed. This requires special glue code on the part of the caller to pick up the return parameters. The hypervisor PFT access routines modify the calling processor's partition PFT on the calling node. On NUMA systems, if an LPAR partition spans multiple Central Electronics Complexes (CECs), the partition's processors may be in separate translation domains. Each platform translation domain has a separate PFT. Therefore, the partition's OS must modify each PFT individually. This is done either by making hcall() accesses specifying the NUMA CEC Cookie (which identifies the translation domain) in the high order 16 bits of the flags parameter (H_ENTER and H_READ only) or by issuing the hcall() from a processor within the translation domain as identified by the processor's NUMA CEC Cookie field of the **"ibm,pft-size"** property.

The PFT is preallocated based upon the value of the partition's PFT_size configuration variable. This configuration variable is initialized to 4 PTEs per node local page frame and 2 PTEs per remote node page frame. The size of the PFT per node is communicated to the partition's OS image via the **"ibm,pft-size"** property of the node.

The value of the configuration variable **PFT_size** consists of two comma separated integers, the first is the number of hardware PFT entries to allocate per CEC local page, and the second is the number of hardware PFT entries to allocate per remote CEC page (if NUMA configured). These allocations are made at partition boot time based upon the initial partition memory allocation, based upon specific situations (such as low page table usage or future need for dynamic memory addition) the OS may wish to override the platform default values.

R1–14.5.4.1–1. For the LPAR option: The platform must allocate the partition's page frame table. The size of this table is determined by the PFT_size configuration variable in the OS image's "common" NVRAM partition.

R1–14.5.4.1–2. For the LPAR option: The platform must provide the **"ibm,pft-size"** property in the processor nodes of the device tree (children of type **cpu** of the **/cpus** node).

Register Linkage (For hcall() tokens 0x04 - 0x18)

- On Call
  - R3 function call token
  - R4 flags (see Table 178, "Page Frame Table Access flags field definition," on page 401)
  - R5 Page Table Entry Index (PTEX)
  - R6 Page Table Entry High word (PTEH) (on H_ENTER only)
  - R7 Page Table Entry Low word (PTEL) (on H_ENTER only)
- On Return:
  - R3 Status Word
  - R4 chosen PTEX (from H_ENTER) / High Order Half of old PTE
  - R5 Low Order Half of old PTE
  - R6

Semantics checks for all hypervisor PTE access routines:

- Hypervisor checks that the caller was in privileged mode or H_Privilege return code.
- On NUMA platforms for the H_ENTER and H_READ calls only, the hypervisor checks that the NUMA CEC Cookie is within the range of values assigned to the partition else return H_Parameter.
- Hypervisor checks that the PTEX is zero or greater and less than the partition maximum, else H_Parameter return code.
- Hypervisor checks the logical address contained in any PTE to be entered into the PFT to insure that it is valid and then translates the logical address into the assigned physical address.
- When hypervisor returns the contents of a PTE, the contents of the RPN are usually architecturally undefined. It is expected that hypervisor implementations leave the contents of this field as it was read from the PTE since it cannot be used by the OS to directly access real memory. The exception to this rule is when the R-XLATE flag is specified to the H_READ hcall(), then the RPN in the PTE is reverse translated into the LPN prior to return.

- Logical addressing:

LPAR adds another level of virtual address translation managed by the hypervisor. The OS is never allowed to use the physical address of its memory this includes System Memory, MMIO space, NVRAM etc. The OS sees System Memory as N regions of contiguous logical memory. Each logical region is mapped by the hypervisor into a corresponding block of contiguous physical memory on a specific node. All regions on a specific system are the same size though different systems with different amount of memory may have different region sizes since they are the quantum of memory allocation to partitions. That is, partitions are granted memory in region size chunks and if a partition's OS gives up memory, it is in units of a full region. On NUMA platforms, groups of regions may be associated with groups of processors forming logical CECs for allocation and migration purposes.

Logical addresses are divided into two fields, the logical region identifier and the region offset. The region offset is the low order bits needed to represent the region size. The logical region identifier are the remaining high order bits.

- Logical addresses start at zero. When control is initially passed to the OS from the platform, the first region is the single RMA. The first region has logical region identifier of zero. This first region is specified by the first address - length pair of the **"reg"** property of the **/memory** node of the OF device tree. Subsequent regions each have their own address - length pair. At initial program load time, the logical region identifiers are sequential starting at zero but over time, with dynamic memory reconfiguration, holes may appear in the partition's address space.
- Logical to physical translation: This translation is based upon a simple indexed table per partition of the physical addresses associated with the start of each region (in logical region identifier order). At least two special values are recognized:

    1. The invalid value for those regions that do not have a physical mapping (so that there can be holes in the logical address map for various reasons such as memory expansion).
    2. The I/O region value, that calls for further checking against partition I/O address range allocations.

    - The logical region identifier is checked for being less than the maximum size, and then used to index the logical to physical translation table.
    - If the physical region identifier is valid (certain values are reserved say 0 and all F's) then it replaces the logical region identifier in the PTE and the PTE access function continues.
    - If the physical region identifier is the I/O region, then proceed to the I/O translation algorithm (implementation dependent based upon platform characteristics).
    - If the physical region identifier is invalid, return H_Parameter

R1–14.5.4.1–3. For the LPAR option: The OS must make no assumptions about the logical to physical mapping other than the low order bits.

R1–14.5.4.1–4. For the LPAR option: Each logical region must have its own address - length pair in the **"reg"** property of the OF **/memory** node.

R1–14.5.4.1–5. For the LPAR option: When control is initially passed to the OS from the platform, the first logical region (having logical region identifier 0) must be the region accessed when the OS operates with translate off.

R1–14.5.4.1–6. For the LPAR option: When control is initially passed to the OS from the platform, the size of the logical region must be equal to a real mode length size supported by the platform.

R1–14.5.4.1–7. For the LPAR option: Each logical region must start and end on a boundary of the largest page size that the logical region supports (see **"ibm,dynamic-memory"** and **"ibm,lmb-page-sizes"** in Appendix B, "LoPAPR Binding," on page 661 as well as R1–14.5.4.1–9 for more details).

R1–14.5.4.1–8. For the LPAR option: The pages that contain the platform's per processor interrupt management areas or any other device marked *"used-by-rtas"* must not be mapped into the partition virtual address space.

R1–14.5.4.1–9. For the LPAR option: Each logical region must support all page sizes presented in the **"ibm,processor-page-sizes"** property in Appendix C, "PA Processor Binding," on page 753 that are less than or equal to the size of the logical region as specified by either the OF standard **"reg"** property of the logical region's OF **/memory** node, or the **"ibm,lmb-size"** property of the logical region's **/ibm,dynamic-reconfiguration-memory** node in Appendix B, "LoPAPR Binding," on page 661.

Implementation Note: 32 bit versions of AIX only support 36 bit logical address memory spaces. Providing such a partition with a larger logical memory address space may cause OS failures.

Implementation Note: Requirement R1–14.5.4.1–7 may be met by ensuring that all logical regions start and end on a boundary of the largest page size supported by the platform.

14.5.4.1.1 H_REMOVE

This hcall is for invalidating an entry in the page table. The PTEX identifies a specific page table entry. If the PFO option is implemented an optional flag causes the hypervisor to compress the page contents to one or more data blocks after invalidating the page table entry given that a compression coprocessor is available and the page is small enough to be synchronously compressed. If the compression coprocessor is busy, or the page is too large, the compression can be subsequently performed using the H_COP_OP hcall() see Section 14.15.2.1, "14.15.2.1 H_COP_OP:," on page 499. If the page contents are compressed, then a checksum may be appended by setting the checksum flag – if the compression flag is not set the checksum flag is ignored.

Syntax:

```
int64                       /* H_Success Expected Return code */
                            /* H_RESCINDED */
                            /* H_Function The compression request is not authorized */
                            /* H_UNSUPPORTED_FLAG */
                            /* H_P3 The out parameter is invalid */
                            /* H_P4 The outlen parameter is invalid */
                            /* H_SG_LIST A scatter/gather list element is invalid */
                            /* H_TOO_BIG The specified page is too long for the output buffer*/
                            /* H_PARTIAL The compression portion of the call was not performed because */

                            /* the compressor was busy */
                            /* H_Constrained The compression portion of the call was not performed because */
                            /* the page was too large to be compressed synchronously */
                            /* H_COP_HW The compressor portion of the call experienced a hardware error */
                            /* H_Busy The hardware is busy user may call back later */
                            /* H_Hardware The hcall() experienced a hardware fault potentially preventing */
                            /* the function */
  hcall     (const uint64 H_REMOVE,
            uint64 flags,   /* see Table 178, "Page Frame Table Access flags field definition," on page 401*/
            uint64 PTEX,    /* The index of the PTE to be removed */
            uint64 AVPN,    /*If the AVPN flag is valid else this parameter value is unused */
                            /*the following two parameters are valid only if the compression flag is on */
            uint64 out,     /*Output data block logical real address */
            int64 outlen    /*If non negative the length of the output data block, */
                            /*If negative the length of the output data descriptor list in bytes *)
            )
```

Parameters:

- flags: AVPN, andcond, and for the CMO option: CMO Option flags as defined in Table 189, "CMO Page Usage State flags Definition," on page 479 and for the PFO option the compression and checksum flags.

- PTEX (index of the PTE in the page table to be used)
- AVPN: Optional “Abbreviated Virtual Page Number” -- used as a check for the correct PTE
  - When the AVPN flag is set, the contents of the AVPN parameter are compared to the first double word of the PTE (after bits 57-63 of the PTE have been masked). Note, the low order 7 bits are undefined and should be zero otherwise the likely result is a return code of H_Not_Found.
  - When the andcond flag is set, the contents of the AVPN parameter are bit anded with the first double word of the PTE. If the result is non-zero the return code is H_Not_Found.
- out: For the PFO option, the output data block logical real address when the compression flag bit is on.
- outlen: For the PFO option, the length of the compression data block or compression data block descriptor list when the compression flag bit is on.

Semantics:

- Check that the PTEX accesses within the PFT else return H_Parameter
- If the AVPN flag is set, and the AVPN parameter bits 0-56 do not match that of the specified PTE then return H_Not_Found.
- If the andcond flag is set, the AVPN parameter is bit anded with the first double word of the specified PTE, if the result is non-zero, then return H_Not_Found.
- The hypervisor Synchronizes the PTE specified by the PTEX and returns its value
  - Use the architected “Deleting a Page Table Entry” sequence such that the first double word of the resultant PFT entry is all 0s.
  - Use the proper tlbie instruction for the page size within a critical section protected by the proper lock (per large page bit in the specified PTE).
  - The synchronized value of the old PTE value ends up in R4 and R5 for return to the caller.
  - For the CMO option: set the page usage state per the CMO Option flags field of the flags parameter as defined in Table 189, “CMO Page Usage State flags Definition,” on page 479.
- For the PFO option: If the Compression flag is on:
  - Check that the calling partition is authorized to use the compression co-processor else return H_Function.
  - If the page is not “main store memory” then return H_UNSUPPORTED_FLAG TBD (value – 312)
  - Check that the page size is <= the compression value in **“ibm,max-sync-cop”** else return H_Constrained.
  - Build CRB for compression of the page size indicated in the PTE
  - If the checksum flag is on command that a checksum be built
  - Verify that the “out” parameter represents a valid logical real address within the caller’s partition else return H_P3
  - If the “outlen” parameter is non-negative verify that the logical real address of (out + outlen) is a valid logical real address within the same 4K page as the “out” parameter else return H_P4.
  - If the “outlen” parameter is negative:
    - Verify that the absolute value of outlen meet all of the follow else return H_P4:
      - Is <= the value of **“ibm,max-sg-len”**

- ❖ Is an even multiple of 16
- ❖ That out + the absolute value of outlen represents a valid logical real address within the same 4K page as the out parameter.

❑ Verify that each 16 byte scatter gather list entry meets all of the following else return H_SG_LIST:

- ❖ Verify that the first 8 bytes represents a valid logical real address within the caller's partition.
- ❖ Verify that the logical real address represented by the sum of the first 8 bytes and the second 8 bytes is a valid logical real address within the same 4K page as the first 8 bytes.

- ■ For the Shared Logical Resource Option if any of the memory represented by the out/outlen parameters have been rescinded then return H_RESCINDED.
- ■ Fill in the destination DDE list from the converted the out/outlen parameters.
- ■ Issue icswx instruction to execute CRB
- ■ Check coprocessor busy – retry / return H_PARTIAL if execution time expired / return H_COP_HW if compressor is broken
- ■ Wait for coprocessor to complete
- ■ If compressor hardware error return H_COP_HW
- ■ Check that the compressor had enough room to house the compressed image else return H_TOO_BIG
- ■ Save compression block size in R6

◆ Return H_Success

14.5.4.1.2 H_ENTER

This hcall adds an entry into the page frame table. PTEH and PTEL contain the new entry. PTEX identifies either the page table entry group or the specific PTE where the entry is to be added, depending upon the setting of the Exact flag. If the Exact flag is off, the hypervisor selects the first free (invalid) PTE in the page table entry group. For pages with sizes less than or equal to 64 K, Flags further provide the option to zero the page, and provide two levels of programmed I-Cache coherence management before activating the page table mapping. This hcall returns the PTE index of the entered mapping. If the PFO option is implemented an optional compression flag causes the hypervisor to initialize the page from one or more compressed data blocks and optionally (checksum flag) check the end to end block data integrity prior to adding the entry to the page table. If the compression flag is not set the checksum flag is ignored.

```
int64                    /* H_Success Expected Return code */
                         /* H_RESCINDED */
                         /* H_TOO_BIG The specified Input stream is too long */
                         /* H_Function The compression request is not authorized */
                         /* H_Aborted The specified input stream was too small to fill the page */
                         /* H_BAD_DATA The initialization data is corrupted */
                         /* H_P4 The in parameter is invalid */
                         /* H_P5 The inlen parameter is invalid */
                         /* H_SG_LIST A Scatter/gather list element is invalid */
                         /* H_Busy The hardware is busy user may call back later */
                         /* PTEX, PTEH, or PTEL parameters are invalid
                         /* H_Hardware The hcall() experienced a hardware fault potentially preventing */
                         /* the function */
    hcall      (const uint64 H_ENTER,
```

```
    uint64   flags,      /* Per Table 178, “Page Frame Table Access flags field definition,” on page 401*/
    int64    PTEX,       /* Index of the first PTE in the page table entry group to be used */
    uint64   PTEH,       /* The high order 8 bytes of the page table entry */
    uint64   PTEL,       /* The low order 8 bytes of the page table entry */
    uint64   in,         /*For the PFO option input data block logical real address */
    int64    inlen       /*For the PFO option if non negative the length of the input data block, */
                         /*If negative the length of the input data descriptor list in bytes *)
)
```

Parameters:

- Flags
  - CEC Cookie
  - Zero Page: Zero the System Memory page in real mode before placing its mapping into the PTE. This flag is ignored for memory mapped I/O space pages; as an attempt to zero missing memory might result in a machine check or worse. This function should use a processor dependent algorithm optimized for maximum performance on the specific hardware. This usually is a sequence of dcbz instructions. Setting this flag for a page with a size larger than 64 K will result in return code of H_TOO_BIG.
  - I-Cache-Invalidate: Issue an icbi etc. instruction sequence to manage the I-Cache coherency of the cachable page. This flag is ignored for memory mapped I/O pages. For use when the D-Cache is known to be clean, before placing its mapping into the PTE. Setting this flag for a page with a size larger than 64 K will result in return code of H_TOO_BIG.
  - I-Cache-Synchronize: Issue dcbst and icbi, etc., instruction sequence to manage the I-Cache coherency of the cachable page. This flag is ignored for memory mapped I/O pages. For use when the D-Cache may contain modified data, before placing its mapping into the PTE. Setting this flag for a page with a size larger than 64 K will result in return code of H_TOO_BIG.
  - Exact: Place the entry in the exact PTE specified by PTEX if it is empty else return H_PTEG_FULL.
  - For the CMO option: CMO Option flags as defined in Table 189, “CMO Page Usage State flags Definition,” on page 479.
  - For the PFO option: the compression flag initializes the page content from a compression buffer and the checksum flag checks for end to end compression buffer data integrity.
- PTEX (index of the first PTE in the page table entry group to be used for the PTE insertion)
- PTEH -- the high order 8 bytes of the page table entry.
- PTEL -- the low order 8 bytes of the page table entry.

Semantics:

- The hypervisor checks that the logical page number is within the bounds of partition allocated memory resources, else returns H_Parameter.
  - If the Shared Logical Resource option is implemented and the logical page number represents a page that has been rescinded by the owner, return H_RESCINDED.
- The hypervisor checks that the address boundary matches the setting of the input PTE’s large page bits; else return H_Parameter.

- The hypervisor checks that the page size described by the setting of the input PTE's page size bits is less than or equal to the largest page size supported by the logical region that is being mapped; else return H_Parameter.
- The hypervisor checks that the WIMG bits within the PTE are appropriate for the physical page number else H_Parameter return. (For System Memory pages WIMG=0010, or, 1110 if the SAO option is enabled, and for IO pages WIMG=01**.)
- For pages with sizes greater than 64 K, the hypervisor checks that the Zero Page, I-Cache-Invalidate, and I-Cache_Synchronize bits of the Flags parameter are not set; else return H_TOO_BIG.
- Force off RS mode reserved PTEL bits (1[1]) as well as hypervisor reserved software bits (57 and 58) in PTEH.
- If the Exact flag is off, set the low order 3 bits of the PTEX to zero (insures that the algorithm stays inside partition's PFT and is faster than a check and error code response).
- If the Zero Page flag is set, use optimized routine to clear page (usually series of dcbz instructions).
- For the PFO option: if the compression flag is on then
  - Check that the calling partition is authorized to use the compression co-processor else return H_Function.
  - If the page is not "main store memory" then return H_UNSUPPORTED_FLAG.
  - Build CRB for decompression
  - If the checksum flag is on command that a checksum be verified.
  - Validate the inlen/in parameters and build the source DDE
    - Verify that the "in" parameter represents a valid logical real address within the caller's partition else return H_P4
    - If the "inlen" parameter is non-negative verify that the logical real address of (in + inlen) is a valid logical real address within the same 4K page as the "in" parameter else return H_P5.
    - If the "inlen" parameter is negative: Verify that the absolute value of inlen meet all of the follow else return H_P5:
      - Is <= the value of **"ibm,max-sg-len"**
      - Is an even multiple of 16
      - That in + the absolute value of inlen represents a valid logical real address within the same 4K page as the in parameter.
    - Verify that each 16 byte scatter gather list entry meets all of the following else return H_SG_LIST:
      - Verify that the first 8 bytes represents a valid logical real address within the caller's partition.
      - Verify that the logical real address represented by the sum of the first 8 bytes and the second 8 bytes is a valid logical real address within the same 4K page as the first 8 bytes.
    - Verify that the sum of all the scatter gather length fields (second 8 bytes of each 16 byte entry) is <= the decompression value in **"ibm,max-sync-cop"** else return H_TOO_BIG.

1.In addition, bits 52 and 53 are forced off on platforms that either do not contain an **"ibm,processor-storage-keys"** property, or contain an **"ibm,processor-storage-keys"** property with the value of zero in both cells. Bit 0 is forced off on platforms that do not have the "Support for the "110" value of the Page Protection (PP) bits" bit set to a value of 1 in the **"ibm,pa-features"** property.

- For the Shared Logical Resource Option if any of the memory represented by the in/inlen parameters have been rescinded then return H_RESCINDED.
- Fill in the source DDE list from the converted the in/inlen parameters.
- Build the destination DDE referencing the start of the PTE page with the length of the PTE page size.
- Issue icswx instruction to execute CRB
- Check coprocessor busy – retry / return H_Busy if execution time exhausted / return H_Hardware if compressor is broken
- Wait for coprocessor to complete
- If compressor ran out of destination space return H_TOO_BIG
- Check that the decompression filled the full page else return H_Aborted
- If the checksum flag is on check that the data is valid else return H_BAD_DATA
- If hardware error return H_Hardware

- If the I-Cache-Invalidate flag is set, issue icbi instructions for all of the page's cache lines
- If the Cache-Synchronize flag is set, issue dcbst and icbi instructions for all of the page's cache lines. Implementations may need to issue a sync instruction to complete the coherency management of the I-Cache.
- The hypervisor selects a PTE within the page table entry group using the following.

Algorithm

- if Exact flag is on then set t to 0 else set t to 7
- for i=0;i<= t; i++
  - Combine page table base, PTEX and offset base on (i) into R3
  - R8 <- ldarx PTEH(R3) /* prepare to take a lock on the PTE */
  - if PTE is valid (R8 (bit 63) is set) then continue
  - if PTE is locked (R8 (bit 57) is set) then continue
  - set R8 (bit 57) /* prepare to lock PTE */
  - PTEH(R3) <- stdcx R8 /* attempt to take lock */
  - if stdcx failed continue
  - goto insert
- return H_PTEG_FULL
- insert: use code sequence from PA Book III
- construct return PTEX (R4 <- (R3 - PFTbase) shifted down 4 places)
- For the CMO option: set the page usage state per the CMO Option flags field of the flags parameter as defined in Table 189, "CMO Page Usage State flags Definition," on page 479.
- return H_Success

14.5.4.1.3 H_READ

This hcall returns the contents of a specific PTE in registers R4 and R5.

int64 hcall (const uint64 H_READ, uint64 flags, int64 PTEX)

Parameters:

- flags:
  - CEC Cookie: Cross CEC PFT access
  - READ_4: Return 4 PTEs
  - R-XLATE: Include a valid logical page number in the PTE if the valid bit is set, else the contents of the logical page number field is undefined.
  - For the CMO option: CMO Option flags as defined in Table 189, “CMO Page Usage State flags Definition,” on page 479.
- PTEX (index of the PTE in the page table to be used -- if the READ_4 flag is set the low order two bits of the PTEX are forced to zero by the hypervisor to insure that they are in the range of the PTEG and it is faster than checking.)

Semantics:

- Checks that the PTEX is within the defined range of the partition’s PFT else return H_Parmaeter
- If the READ_4 bit is clear Then load the specified PTE into R4 and R5
  - If R-XLATE flag is set, then reverse translate the RPN field into the logical page number.
- Else
  - clear the two low order bits of the PTEX (faster than checking them)
  - load the 4 PTEs starting at PTEX into R4 through R11.
    - If R-XLATE flag is set, then reverse translate the RPN fields into the logical page number.
- For the CMO option: set the page usage state per the CMO Option flags field of the flags parameter as defined in Table 189, “CMO Page Usage State flags Definition,” on page 479.
- Set H_Success in R3 and return

14.5.4.1.4 H_CLEAR_MOD

This hcall clears the modified bit in the specific PTE. The second double word of the old PTE is returned in R4.

```
int64 /* H_Success                  Expected Return Code */
      /* H_PARAMETER                The PTE index was out of bounds */
      /* H_Not_Found                The requested valid PTE Entry was not found */
      hcall (   const uint64 H_CLEAR_MOD, /* Clears the PTE Modified bit */
                uint64 flags,             /* None Defined */
                int64 PTEX)               /* Index of the PTE to be used */
```

Parameters:

- flags: For the CMO option: CMO Option flags as defined in Table 189, “CMO Page Usage State flags Definition,” on page 479.
- PTEX (index of the PTE in the page table to be used)

Semantics:

- Check that the PTEX accesses within the PFT, else returns H_Parameter
- Check that the “V” bit is one, else return H_Not_Found.
- Fetch the low order double word of the PTE into R4. If the “C” bit is zero, then return H_Success.
- The hypervisor synchronizes the PTE specified by the PTEX, clears the mod bit, and returns its old value:
  - Use the architected “Modifying a Page Table Entry General Case” sequence from PA Book III.
  - Only PTE bits to be modified are:
    - In double word 0 SW bit 57 and the V bit (63)
    - In double word 1, C bit (56).
  - Use the proper tlbie instruction for the page size (per large page flag within PTE) within a critical section protected by the proper lock.
  - The second double word of the old PTE value ends up in R4.
  - At the point where the new values are to be activated, use the old values with the “C” bit cleared.
  - For the CMO option: set the page usage state per the CMO Option flags field of the flags parameter as defined in Table 189, “CMO Page Usage State flags Definition,” on page 479.
  - Return H_Success

14.5.4.1.5 H_CLEAR_REF

This hcall clears the reference bit in the specific PTE. The second double word of the old PTE is returned in R4.

```
int64 /* H_Success                  Expected Return Code */
      /* H_Parameter                The PTE index was out of bounds */
      /* H_Not_Found                The requested valid PTE Entry was not found */
      hcall (   const uint64 H_CLEAR_REF, /* Clears the PTE Reference bit */
                uint64 flags,             /* None Defined */
                int64 PTEX)               /* index of the PTE to be used */
```

Parameters:

- flags: For the CMO option: CMO Option flags as defined in Table 189, “CMO Page Usage State flags Definition,” on page 479.
- PTEX (index of the PTE in the page table to be used)

Semantics:

- Check that the PTEX accesses within the PFT, else return H_Parameter.
- Check that the “V” bit is one, else return H_Not_Found.

- Only PTE bits to be modified are:
  - In double word 1 the R bit (55)
- Use the architected “Resetting the Reference Bit” sequence from PA Book III with the original second double word of the PTE ending up in R4.
- For the CMO option: set the page usage state per the CMO Option flags field of the flags parameter as defined in Table 189, “CMO Page Usage State flags Definition,” on page 479.
- Return H_Success

14.5.4.1.6 H_PROTECT

This hcall sets the page protect bits in the specific PTE.

```
int64 /* H_Success                    Expected Return Code */
      /* H_Parameter                  The PTE index was out of bounds */
      /* H_Not_Found                  The requested valid PTE Entry was not found */
      hcall (   const uint64 H_PROTECT, /* Changes the page protection specification */
                uint64 flags,           /* Special function indications */
                int64 PTEX,             /* index of the PTE to be used */
                uint64 AVPN)            /* Abbreviated virtual page number */
```

Parameters:

- flags: AVPN, pp0[1], pp1, pp2, key0-key4[2], n, and for the CMO option: CMO Option flags as defined in Table 189, “CMO Page Usage State flags Definition,” on page 479.
- PTEX (index of the PTE in the page table to be used)
- AVPN: Optional “Abbreviated Virtual Page Number” -- used as a check for the correct PTE when the AVPN flag is set.

Semantics:

- Check that the PTEX accesses within the PFT, else return H_Parameter
- Check that the “V” bit is one, else return H_Not_Found.
- If the AVPN flag is set, and the AVPN parameter bits 0-56 do not match that of the specified PTE, then return H_Not_Found.
- The hypervisor synchronizes the PTE specified by the PTEX, sets the pp0[3], pp1, pp2, key0-key4[4], and n bits per the flags parameter.
  - Only PTE bits to be modified are:

1.The pp0 portion of the flags parameter is ignored on platforms that do not have the “Support for the “110” value of the Page Protection (PP) bits” bit set to a value of 1 in the **“ibm,pa-features”** property.

2.The key0-key4 portion of the flags parameter is ignored on platforms that either do not contain an **“ibm,processor-storage-keys”** property, or contain an **“ibm,processor-storage-keys”** property with the value of zero in both cells.

3.The pp0 bit is not modified on platforms that do not have the “Support for the “110” value of the Page Protection (PP) bits” bit set to a value of 1 in the **“ibm,pa-features”** property.

4.The key0 - key4 bits are not modified on platforms that either do not contain an **“ibm,processor-storage-keys”** property, or contain an **“ibm,processor-storage-keys”** property with the value of zero in both cells.

- ❖ In double word 0 SW bit 57 and the V bit (63)
- ❖ In double word 1 pp0 (see footnote 3 on page 416), pp1, pp2, key0-key4 (see footnote 4 on page 416), and n

- ❑ Use the architected “Modifying a Page Table Entry General Case” sequence.
- ❑ Use the proper tlbie instruction for the page size (per value in PTE) within a critical section protected by the proper lock.
- ❑ At the point where the new values are to be activated use the old values with the “R” bit cleared and the pp0 (see footnote 3 on page 416), pp1, pp2, key0-key4 (see footnote 4 on page 416), and n bits set as specified in the flags parameter.
- ❑ For the CMO option: set the page usage state per the CMO Option flags field of the flags parameter as defined in Table 189, “CMO Page Usage State flags Definition,” on page 479.
- ❑ Return H_Success

14.5.4.1.7 H_BULK_REMOVE

This hcall is for invalidating up to four entries in the page table. The PTEX in the translation specifier high parameters identifies the specific page table entries.

Prototype:

```
int64                   /* H_Success, expected return code
                        H_HARDWARE, Hardware Error
                        H_PARAMETER, One or more parameters were in error first found was flagged */
    hcall (   const int64 H_BULK_REMOVE,    /* Function Code */
              uint64 TSH1                   /*Translation Specifier High 1*/
              uint64 TSL1                   /*Translation Specifier Low 1*/
              uint64 TSH2                   /*Translation Specifier High 2*/
              uint64 TSL2                   /*Translation Specifier Low 2*/
              uint64 TSH3                   /*Translation Specifier High 3*/
              uint64 TSL3                   /*Translation Specifier Low 3*/
              uint64 TSH4                   /*Translation Specifier High 4*/
              uint64 TSL4);                 /*Translation Specifier Low 4*/
```

Translation specifiers:

Each is 16 bytes long made up of two 8 byte double words; a translation specifier high and a translation specifier low.

- Translation Specifier High double word:
  - First byte (0) is a control/status byte:
    - ❑ High order two bits (0 and 1) are type code:
      - ❖ 00 Unused -- if found stop processing and return H_PARAMETER
      - ❖ 01 Request -- Processes As per H_REMOVE as modified by low order two control bits.
      - ❖ 10 Response -- written by hypervisor as a return status from processing individual “request” translation specifier
      - ❖ 11 End of String -- if found stop processing and return H_Success.
    - ❑ Next two bits (2 and 3) are response code (in response to processing an individual “request” translation specifier (type code modified to 10)):

- ❖ 00 Success -- the specified translation was removed as per H_REMOVE with the PTE's RC bits in the next two status bits.
- ❖ 01 Not found -- the specified translation was not found as per H_REMOVE.
- ❖ 10 H_PARM -- one or more of the parameters of the specified translation were invalid per H_REMOVE -- processing of the bulk entries stops at this point and the hypervisor returns H_PARAMETER.
- ❖ 11 H_HW -- The hardware experienced an uncorrected error processing this translation specifier -- processing of the bulk entries stops at this point and the hypervisor returns H_HARDWARE.

❑ Next two bits (4 and 5) are the Reference/Change bits from the removed PTE (These bits are only valid if bits 0-3 are 1000):

❑ Low order two bits (6 and 7) are request modification flags:

- ❖ 00 absolute -- remove the specified PTEX entry unconditionally
- ❖ 01 andcond -- remove the specified PTEX entry as with the andcond flag of H_REMOVE
- ❖ 10 AVPN -- remove the specified PTEX entry as with the AVPN flag of H_REMOVE
- ❖ 11 not used -- if found stop processing and return H_PARAMETER.

■ Bytes 1 through 7 are the PTEX (PFT byte offset divided by 16)

◆ Translation Specifier Low double word:

■ Bytes 0 through 7 are the AVPN as per H_REMOVE

Semantics:

◆ For each translation specifier, while the translation specifier is not “end of string”:

■ Check that the PTEX accesses within the PFT else set H_PARM response status in the specific translation specifier high register and return H_Parameter

■ If the AVPN flag is set, and the AVPN parameter bits 0-56 do not match that of the specified PTE then set response status Not found in the specific translation specifier high register, Continue.

■ If the andcond flag is set, the AVPN parameter is bit anded with the first double word of the specified PTE (after bits 57-63 of the PTE have been masked), if the result is non-zero, then set response status Not found in the specific translation specifier high register, Continue. (Note the low order 7 bits of the AVPN parameter should be zero otherwise the likely result is a response status of Not found).

■ The hypervisor Synchronizes the PTE specified by the PTEX.

■ Use the architected “Deleting a Page Table Entry” sequence.

■ Use the proper tlbie instruction for the page size within a critical section protected by the proper lock (per large page bit in the specified PTE).

■ The synchronized value of the old PTE RC bits ends up in bits 4 and 5 of the individual translation specifier high register along with success response status.

◆ return H_Success

14.5.4.2 Translation Control Entry Access

The Translation Control Entry (TCE) access hcall()s take as a parameters the Logical I/O Bus Number (LIOBN) that is the logical bus number value derived from the **"ibm,dma-window"** property associated with the particular IOA. For the format of the **"ibm,dma-window"** property, reference Appendix B, "LoPAPR Binding," on page 661.

14.5.4.2.1 H_GET_TCE

This hcall() returns the contents of a specified Translation Control Entry.

Syntax:

```
int64                   /* H_Success:      Expected Return Code */
                        /* H_PARTIAL:    RPN not relative to callers logical address space */
                        /* H_PARAMETER:          LIOBN or IOBA out of range */
                        /* H_RESCINDED: A specified parameter refers to a rescinded shared logical resource/
                        /* H_Hardware:   The hardware experienced a fault causing the function to fail. */
    hcall (const uint64 H_GET_TCE,    /* Return the contents of the specified TCE */
                        uint32 LIOBN, /* Logical I/O Bus Number for TCE table to be accessed */
                        uint64 IOBA)  /* I/O Bus Address for indexing into the TCE table */
```

Parameters:

- LIOBN (Logical I/O Bus Number for TCE table to be accessed)
- IOBA (I/O Bus Address for indexing into the TCE table)

Semantics:

- If the LIOBN, or IOBA are outside of the range of calling partition assigned values return H_PARAMETER.
  - If the Shared Logical Resource option is implemented and the LIOBN, or IOBA represents a logical resource that has been rescinded by the owner, return H_RESCINDED.
- Load R4 with the specified TCE contents.
- If specified TCE's Page Mapping and Control bits (see Section 3.2.2.2, "DMA Address Translation and Control via the TCE Mechanism," on page 65) specify "Page Fault" then return H_Success
- Reverse translate the TCE's RPN field into its logical page number
- If the logical page number is owned by the calling partition then replace the RPN field of R4 with the logical page number and return H_Success.
- Logically OR the contents of R4 with 0xFFFFFFFFFFFFF000 placing the result into R4.
- Return H_PARTIAL.

14.5.4.2.2 H_PUT_TCE

This hcall() enters the mapping of a single 4 K page into the specified Translation Control Entry.

Syntax:

```
int64                   /* H_Success: Expected return code */
                        /* H_Parameter: One or more of the parameters were out of range */
                        /* H_RESCINDED: A specified parameter refers to a rescinded shared logical resource/
                        /* H_Hardware: The function failed due to unrecoverable hardware error */
    hcall      (const uint64 H_PUT_TCE,        /* Function Token */
```

```
                uint32 LIOBN,     /* Logical I/O Bus Number of TCE table to be accessed (from dma window property)
*/
                uint64 IOBA,      /* I/O Bus Address for indexing into TCE table */
                uint64 TCE);      /* TCE contents to be stored in the TCE table (contains logical address of storage
page to be mapped*/
```

Semantics:

- If the LIOBN or IOBA parameters are outside of the range of calling partition assigned values return H_PARAMETER.
  - If the Shared Logical Resource option is implemented and the LIOBN, or IOBA represents a logical resource that has been rescinded by the owner, return H_RESCINDED.
- If the Page Mapping and Control field of the TCE is not “Page Fault” (see Section 3.2.2.2, “DMA Address Translation and Control via the TCE Mechanism,” on page 65)
  - Then if the logical address within the TCE parameter is outside of the range of calling partition assigned values
    - Then return H_PARAMETER.
    - Else translate the logical address within the TCE parameter into the corresponding physical real address.
- The hypervisor stores the TCE resultant value in the TCE table specified by the LIOBN and IOBA parameters; returning H_Success. (In the “Page Fault” case the RPN remains untranslated.)

Software Note: The PA requires the OS to issue a sync instruction to proceed the signalling of an IOA to start an IO operation involving DMA to guarantee the global visibility of both DMA and TCE data. This hcall() does not include a sync instruction to guarantee global visibility of TCE data and in no way diminishes the requirement for the OS to issue it.

14.5.4.2.3 H_STUFF_TCE

This hcall() duplicates the mapping of a single 4 K page through out a contiguous range of Translation Control Entries. Thus, in initializing and/or invalidating many entries. To retain interrupt responsiveness this hcall() should be called with a count parameter of no more than 512, LoPAPR architecture provides enforcement for this restriction to aid in client code debug.

Syntax:

```
int64                   /* H_Success: Expected return code */
                        /* H_Parameter: One or more of the parameters were out of range */
                        /* H_RESCINDED: A specified parameter refers to a rescinded shared logical resource
                        /* H_P4: The count parameter is greater than 512 */
                        /* H_Hardware: The function failed due to unrecoverable hardware error */
    hcall       (const uint64 H_STUFF_TCE,      /* Function Token */
                uint32 LIOBN,     /* Logical I/O Bus Number of TCE table to be accessed (from dma window prop.) */
                uint64 IOBA,      /* The starting I/O Bus Address for indexing into TCE table */
                uint64 TCE,       /* TCE contents to be stored in the TCE table (contains logical address of storage */
                                  /* page to be mapped*/
                uint64 count);    /* The number of consecutive TCEs to fill */
```

Semantics:

- If the LIOBN, or IOBA, are outside of the range of calling partition assigned values return H_PARAMETER.
  - If the Shared Logical Resource option is implemented and the LIOBN, or IOBA represents a logical resource that has been rescinded by the owner, return H_RESCINDED.

- If the count parameter is greater than 512 then return H_P4
- If the count parameter added to the TCE index specified by IOBA is outside of the range of the calling partition assigned values return H_PARAMETER.
- If the Page Mapping and Control field of the TCE is not “Page Fault” (see Section 3.2.2.2, “DMA Address Translation and Control via the TCE Mechanism,” on page 65)
  - Then if the logical address within the TCE parameter is outside of the range of calling partition assigned values
    - Then return H_PARAMETER.
      - If the Shared Logical Resource option is implemented and the logical address’s page number represents a page that has been rescinded by the owner, return H_RESCINDED.
    - Else translate the logical address within the TCE parameter into the corresponding physical real address.
- The hypervisor stores the TCE resultant value in the TCE table entries specified by the LIOBN, IOBA and count parameters; returning H_Success. (In the “Page Fault” case the RPN remains untranslated.)

Implementation Note: The PA requires the OS to issue a sync instruction to proceed the signaling of an IOA to start an IO operation involving DMA to guarantee the global visibility of both DMA and TCE data. This hcall() does not include a sync instruction to guarantee global visibility of TCE data and in no way diminishes the requirement for the OS to issue it.

14.5.4.2.4 H_PUT_TCE_INDIRECT

This hcall() enters the mapping of up to 512 4 K pages into the specified Translation Control Entry. The LIOBN parameter if positive is the cookie (LIOBN) of the specific TCE table to load. For the Multi-TCE Table (MTT) option, if the LIOBN parameter is negative, CNT = the absolute value of LIOBN (up to 128), and the first CNT 8 byte entries of the buffer referenced by the TCE parameter contains the TCE table cookies (LIOBNs) for the various TCE tables to load (up to a maximum of 128 TCE tables).

Note: Users of the MTT option that are subject to partition migration should be prepared for the loss of support for the MTT option after partition migration.

Syntax:

```
int64                   /* H_Success: Expected return code */
                        /* H_Parameter: One or more of the parameters were out of range */
                        /* H_RESCINDED: A specified parameter refers to a rescinded shared logical resource/
                        /* H_Function: The functional extension to multiple LIOBNs is not enabled */
                        /* H_Hardware: The function failed due to unrecoverable hardware error */
    hcall   (const uint64 H_PUT_TCE_INDIRECT,       /* Function Token */
             int64 LIOBN,       /* Logical I/O Bus Number of TCE table to be accessed (from dma window property)
                                   or if negative, the number of LIOBN fields in the buffer*/
             uint64 IOBA,       /* The starting I/O Bus Address for indexing into TCE table */
             uint64 TCE,        /* The logical address of a page of (4 K long on a 4 K boundary) of TCE contents to
be stored in the TCE table (contains logical address of storage page to be mapped)*/
             uint64 count);     /* The number of consecutive TCEs to fill */
```

Semantics:

/* Validate the input parameters */

- If the LIOBN parameter is non-negative then do
  - If the count parameter is > 512 then return H_Parameter.

- If the Shared Logical Resource option is implemented and the LIOBN parameter represents a TCE table that has been rescinded by the owner, return H_RESCINDED.
- If the LIOBN parameter represents a TCE table that is not valid for the calling partition, return H_Parameter.
- Liobns[0] = the LIOBN parameter.
- If the Shared Logical Resource option is implemented and any of the I/O bus address range represented the IOBA parameter plus count pages within the TCE table represented by the LIOBN parameter represents rescinded resource, return H_RESCINDED.
- If any of the I/O bus address range represented the IOBA parameter plus count pages within the TCE table represented by the LIOBN parameter is not valid for the calling partition then return H_Parameter.
- end

- Else do
  - If the MTT Option is not enabled return H_Function.
  - If the LIOBN parameter < -128 then return H_Parameter.
  - If the sum of the count parameter plus |LIOBN| is > 512 then return H_Parameter.
  - end
- If the Shared Logical Resource option is implemented and the TCE parameter represents a logical page address of a page that has been rescinded by the owner, return H_RESCINDED.
- If the TCE parameter represents the logical page address of a page that is not valid for the calling partition, return H_Parameter.
- Copy the contents of the page referenced by the TCE table to a temporary hypervisor page (Temp) for validation without the potential for caller manipulation.

/* Validate the indirect parameters */

- VAL= 0
- If the LIOBN parameter is negative then do
  - For CNT = 1,|LIOBN|,1
    - T = 8 byte entry Temp [VAL]
    - If the Shared Logical Resource option is implemented and T as an LIOBN represents a TCE table that has been rescinded by the owner, return H_RESCINDED.
    - If T as an LIOBN represents a TCE table that is not valid for the calling partition, return H_Parameter.
    - Liobns[VAL+] = T.
    - If the Shared Logical Resource option is implemented and any of the I/O bus address range represented the IOBA parameter plus count pages within the TCE table represented by “T” as an LIOBN represents a rescinded resource, return H_RESCINDED.
    - If any of the I/O bus address range represented the IOBA parameter plus count pages within the TCE table represented by “T” as an LIOBN is not valid for the calling partition then return H_Parameter.
    - loop

- end

/* Translate the logical pages addresses to physical*/

- for CNT = 1,count,1
  - T = 8 byte entry Temp [VAL+]
  - If the Page Mapping and Control field of the 8 byte entry “T” is not “Page Fault” (see Table 8, “TCE Definition,” on page 66) then do
    - If the Shared Logical Resource option is implemented and the value of “T” as a logical address represents a page that has been rescinded by the owner, then return H_RESCINDED.
    - If “T” as a logical address is outside of the range of calling partition assigned values then return H_PARAMETER.
    - Translate the logical address within the TCE buffer entry into the corresponding physical real address.
    - Temp[CNT – 1] = translated physical real address.
    - end
  - loop

/* Fill the TCE table(s) */

- If LIOBN parameter is negative then VAL = |LIOBN| else VAL = 1.
- For TABS = 1, VAL, 1
  - The TCE table to fill is that referenced by Liobns[VAL] as an LIOBN.
  - INDEX = the page index within the TCE table represented by the IOBA parameter.
  - For CNT = 1, count, 1
    - TCE_TABLE [Liobns[VAL], INDEX+] = Temp [CNT-1]
    - Loop
  - Loop
- Return H_Success.

Implementation Note: The PA requires the OS to issue a sync instruction to proceed the signaling of an IOA to start an IO operation involving DMA to guarantee the global visibility of both DMA and TCE data. This hcall() does not include a sync instruction to guarantee global visibility of TCE data and in no way diminishes the requirement for the OS to issue it.

14.5.4.3 Processor Register Hypervisor Resource Access

Certain processor registers are architecturally hypervisor resources, in the following cases the hypervisor provides controlled write access services.

14.5.4.3.1 H_SET_SPRG0

int64 hcall (const uint64 H_SET_SPRG0, uint64 value)

Parameters:

- value: The value to be written into SPRG0. No parameter checking is done against this value.

14.5.4.3.2 H_SET_DABR

Note: Implementations reporting compatibility to ISA versions less than 2.07 are required to implement this interface; however, this interface is being deprecated in favor of "H_SET_MODE" on page 425 for newer implementations.

int64 hcall (const uint64 H_SET_DABR, uint64 value)

Semantics:

- If the platform does not implement the extended DABR facility then:
  - Validate the value parameter else return H_RESERVED_DABR and the value in the DABR is not changed:
    - The DABR BT bit (Breakpoint Translation) is checked for a value of 1.
- Else (The platform does implement the extended DABR facility):
  - Load the DABRX register with 0b0011.
  - place the value parameter into the DABR.
- Return H_Success.

14.5.4.3.3 H_PAGE_INIT

int64 hcall (const uint64 H_PAGE_INIT, uint64 flags, addr64 destination, addr64 source)

Parameters:

- flags: zero, copy, I-Cache-Invalidate, I-Cache-Synchronize, and for the CMO option: CMO Option flags as defined in Table 189, "CMO Page Usage State flags Definition," on page 479.
- destination: The logical address of the start of the page to be initialized
- source: The logical address of the start of the page use as the source on a page copy initialization. This parameter is only checked and used if the copy flag is set.

Semantics:

- The logical addresses are checked, they must both point to the start of a 4 K system memory page associated with the partition or return H_Parameter.
  - If the Shared Logical Resource option is implemented and the source/destination logical page number represents a page that has been rescinded by the owner, return H_RESCINDED.
- If the zero flag is set, clear the destination page using a platform specific routine (usually a series of dcbz instructions).
- If the copy flag is set, execute a platform specific optimized copy of the full 4 K page from the source to the destination.
- If I-Cache-Invalidate flag is set, issue icbi instructions for all of the page's cache lines
- If I-Cache-Synchronize flag is set, issue dcbst and icbi instructions for all of the page's cache lines. Implementations may need to issue a sync instruction to complete the coherency management of the I-Cache.

- For the CMO option: set the page usage state per the CMO Option flags field of the flags parameter as defined in Table 189, “CMO Page Usage State flags Definition,” on page 479.
- Return H_Success

Note: For the CMO option, the CMO option flags may be used to notify the platform of the page usage state of a page without regard to its hardware page table entry or lack there of independent of any other option flags.

14.5.4.3.4 H_SET_XDABR

Note: Implementations reporting compatibility to ISA versions less than 2.07 are required to implement this interface; however, this interface is being deprecated in favor of “H_SET_MODE” on page 425 for newer implementations.

This hcall() provides support for the extended Data Address Breakpoint facility. It sets the contents of the Data Address Breakpoint Register (DABR) and its companion Data Address Breakpoint Register Extension (DABRX). A principal advantage of the extended DABR facility is that it allows setting breakpoints for LPAR addresses that the hypervisor had to preclude using the previous facility.

```
int64 hcall                          /* H_Success: Expected Return Code */

                                     /* H_Hardware hardware experienced an unrecoverable error */
                                     /* H_Parameter invalid parameter value */
        (const uint64 H_SET_XDABR, /*Function Token */
        uint64 value,                /* Value to be placed in DABR register */
        uint64 extended);            /* Value to be place in DABRX register */
```

Semantics:

- Validates the extended parameter else return H_Parameter:
  - Reserved Bits (0-59) are zero.
  - The HYP bit (61) is off.
  - The rest of the PRIVM field (Bits 62-63) is one of those supported:
    - 0b01 Problem State
    - 0b10 Privileged non-hypervisor
    - 0b11 Privileged or Problem State
    - (Specifying neither Problem or Privileged state is not supported)
- Load the validated extended parameter into the DABRX
- Load the value parameter into the DABR
- Return H_Success.

14.5.4.3.5 H_SET_MODE

This hcall() is used to set hypervisor processing resource mode registers such as breakpoints and watchpoints. The modes supported by the hardware processor are a function of the processor architectural level as reported in the **`“cpu-version”`** property. Table 180, “H_SET_MODE Parameters per ISA Level,” on page 427 presents the valid parameter ranges for the architectural level reported in the **`“cpu-version”`** property.

Setting breakpoints: A breakpoint is set for a hardware tread. Should the hardware thread complete an instruction who's effective address matches that of the set breakpoint a trace interrupt is signaled. When setting the breakpoint resource, the mflags and value2 parameters are zero. The value1 parameter is the effective address of the breakpoint to be set.

Setting watchpoints: A watchpoint is set for a hardware tread. Should the hardware thread attempt to access within the specified double word range of the effective address specified by the value1 parameter as qualified by the conditions specified in the value2 parameter a Data Storage type interrupt is signaled. When setting the watchpoint resource, the mflags parameter is zero. The value1 parameter is the effective double word address of the watchpoint to be set. The value2 parameter specifies the qualifying conditions for the access, these are a subset of the POWER ISA conditions that are relevant within the context of a logical partition. This subset includes the MRD field, DW, DR, WT, WTI, PNH, and PRO bits. All other value2 fields are zero.

Setting Interrupt Vector Location Modes: The Alternate Interrupt Location (AIL) Mode for the calling partition is set. Since this function has partition wide scope, it may take longer for the hypervisor to perform the function on all processors than is permissible during a synchronous call; therefore, the call might return long busy. In that case the caller should repeat the call with the same parameters after the specified time period until the H_SUCCESS return code is received. A call with different parameters indicates the beginning of a new partition wide mode setting. The desired AIL mode is encoded in the two low order mflags bits (all other mflags bits are 0) while both value1 and value2 parameters are zero.

```
int64                                                   / * H_Success: Expected Return Code * /
                                                        / * H_UNSUPPORTED_Flag invalid mflags bit * /
                                                        / * H_P2 invalid resource encoding * /
                                                        / * H_P3 invalid value1 * /
                                                        / * H_P4 invalid value2 * /
                                                        / * H_LongBusyOrder10mSec not done yet * /
        hcall   (const uint64 H_SET_MODE,               /* Set the mode of the specified processing resource */
                                                        /* per the specified value*/
                uint64  mflags,                         /* Processing resource specific flags*/
                uint64  resource,                       /* Processing resource identifier */
                uint64  value1,                         /* Value(s) to set the resource */
                uint64  value2)
```

Semantics:

```
switch (resource) {
    case 0: /* not used /
            return H_P2;
            break;
    case 1: /* Completed Instruction Address Breakpoint Register */
            if value2 <> 0 the return H_P4;
            if mflags <> 0 then return H_UNSUPPORTED_FLAG;
            If low order two bits of value1 are 0b11 then return H_P3; /* not hypervisor instruction address */
            move value 1 into CIABR; /* note the value2 parameter is not used for this resource */
            break;
    case2: /* Watch point 0 registers */
            if mflags <> 0 then return H_UNSUPPORTED_FLAG;
            If value2 bit 61 == 0b1 then return H_P4; /* not hypervisor addresses */
            move value1 into DAWR0;
            move value2 into DAWRX0;
            break;
    case3: / * Address Translation Mode on Interrupt * /
            if value1 < > 0 then return H_P3;
            if value2 < > 0 then return H_P4;
            switch (mflags) {
                    case 0: / * IR = DR = 0 * /
                            Set LPCR AIL field of calling partition processors to 0b00;
                            break;
```

```
                case 1:  / * reserved * /
                         return H_UNSUPPORTED_FLAG ( - 318);
                         break;
                case 2:  / * IR = DR = 1 interrupt vectors at E.A. 0X18000 * /
                         Set LPCR AIL field of calling partition processors to 0b10;
                         break;
                case 3:  / * IR = DR = 1 interrupt vectors at E.A. 0XC000 0000 0000 4000 * /
                         Set LPCR AIL field of calling partition processors to 0b11;
                         break;
                default: return H_UNSUPPORTED_FLAG (value based on most convenient
                         unsupported bit);
                         break;
default:
        return H_P2;
        break; }
```

Table 180. H_SET_MODE Parameters per ISA Level

ISA level	Supported Resource Values	Values Supported mflags	Value 1	Value 2	Comments
2.07	1	None	Breakpoint Address	None	
	2	None	Watchpoint Double Word Address	Watchpoint Qualifying Conditions	
	3	0	None	None	IR=DR=0 No offset
		1	None	None	Reserved
		2	None	None	IR=DR=1 offset 0x18000
		3	None	None	IR=DR=1 offset 0xC000 0000 0000 4000
		All Others	All Others	All Others	Reserved
	All Others	All Others	All Others	All Others	Reserved

R1–14.5.4.3.5–1. For implementations supporting POWER ISA level 2.07 and beyond: the platform must implement the H_SET_MODE hcall() per the syntax and semantics of section 14.5.4.3.5, “H_SET_MODE,” on page 425.

14.5.4.4 Debugger Support hcall()s

The real mode debugger needs to be able to get to its async port and beyond the real mode limit register without turning on virtual address translation. The following hcall()s provide that capability.

14.5.4.4.1 H_LOGICAL_CI_LOAD

int64 hcall (const uint64 H_LOGICAL_CI_LOAD, uint64 size, uint64 addr)

Parameters:

- size: The size of the cache inhibited load:
  - byte = 1
  - half = 2
  - full = 4
  - double=8
  - All other size values are illegal and returns H_Parameter
- addr: The logical address of the cache inhibited location to be read. The hypervisor checks that the address is within the range of addresses valid for the partition, on a boundary equal to the requested length, is not to the location BA+4 within an interrupt management area, and mapped as cache inhibited (cache paradoxes are to be avoided)-- Else H_Parameter.
- On successful return (H_Success), the read value is low order justified in register R4.

14.5.4.4.2 H_LOGICAL_CI_STORE

int64 hcall (const uint64 H_LOGICAL_CI_STORE, uint64 size, uint64 addr, uint64 value)

Parameters:

- size: The size of the cache inhibited store:
  - byte = 1
  - half = 2
  - full = 4
  - double=8
  - All other size values are illegal and returns H_Parameter
- addr: The logical address of the cache inhibited location to be written. The hypervisor checks that the address is within the range of addresses valid for the partition, on a boundary equal to the requested length, is not to the location BA+4 within an interrupt management area, and mapped as cache inhibited (cache paradoxes are to be avoided).
- value The value to be written is low order justified in register R6.

14.5.4.5 Virtual Terminal Support

This section has been moved to Section 16.6, “Virtual Terminal (Vterm),” on page 582.

Architecture and Implementation Note: The requirement to provide the **`“ibm,termno”`** property in the **`/rtas`** node, has been removed (it is now necessary to look for **`vty`** nodes and use their unit address from the **`“reg”`**

property to get the same information). The **"ibm,termno"** property called for sequential terminal numbers, but with the use of unit addresses from the **"reg"** property, such is not the case.

14.5.4.6 Dump Support hcall()s

To allow the OS to dump hypervisor data areas in support of field problem diagnosis the hcall-dump support function set contains the H_HYPERVISOR_DATA hcall(). This hcall() is enabled or disabled (default disabled) via the Hardware Management Console. If the hcall-dump function set is disabled an attempt to make a H_HYPERVISOR_DATA hcall() returns H_Function. When the function is enabled, the hcall-dump function set is specified in the **"ibm,hypertas-functions"** property. The requester calls repeatedly starting with a control value of zero getting back 64 bytes per call and setting the control parameter on the next call to the previous call's return code until the hcall() returns H_Parameter indicating that all hypervisor data has been dumped. The precise meaning of the sequence of data is implementation dependent. The H_HYPERVISOR_DATA hcall() need only return data in the firmware working storage that is not contained in the PFT or TCE tables since the contents of these tables are available to the OS.

14.5.4.6.1 H_HYPERVISOR_DATA

int64 hcall (const uint64 H_HYPERVISOR_DATA, uint64 control)

Parameters:

- control: A value passed to establish the progress of the dump.

Semantics:

- If the control value is zero, the data returned is the first segment of the hypervisor's working storage, with a non-negative return code.
- If the control value is equal to the return code of the last H_HYPERVISOR_DATA call, and the return code is non-negative, the data returned in R4 through R11 is the next sequential segment of the hypervisor's working storage. The contents of R4 through R11 are undefined if the return code is negative.

Implementation Note: It is expected that the control value is be used by the H_HYPERVISOR_DATA routine as an offset into the hypervisor's data area. For the expected implementation, hypervisor checks the value of the control parameter to insure that the resultant pointer is within hypervisor's data area else it returns H_Parameter.

14.5.4.7 Interrupt Support hcall()s

Injudicious values written to the interrupt source controller may affect innocent partitions. The following hcall()s monitor the architected functions.

14.5.4.7.1 H_EOI

Software Implementation Note: Issuing more H_EOI calls than actual interrupts may cause undesirable behavior, including but not limited to lost interrupts, and excessive phantom interrupts.

int64 hcall (const uint64 H_EOI, uint64 xirr)

Parameters:

- xirr: The low order 32 bits is the value to be written into the calling processor's interrupt management area's external interrupt request register (xirr).

Semantics:

- If the platform implements the Platform Reserved Interrupt Priority Level Option, and the priority field of the xirr parameter matches one of the reserved interrupt priorities then return H_Resource.

- If the value of the xirr parameter is such that the low order 3 bytes (xisr) is one of the interrupt source values assigned to the partition, and the high order byte xirr byte (cppr) is equal or less favored than the current cppr contents, then the value is written into the calling processor's xirr causing the interrupt source controller to signal an "end of interrupt" (EOI) to the specified interrupt source logic, then hypervisor returns H_Success or H_Hardware (if an unrecoverable hardware error occurred). If the xirr value is not legal, hypervisor returns H_Parameter.

  - If the Shared Logical Resource option is implemented and the xirr parameter represents a shared logical resource location that has been rescinded by the owner, return H_RESCINDED.

14.5.4.7.2 H_CPPR

int64 hcall (const uint64 H_CPPR, uint64 cppr)

Parameters:

- cppr: The low order byte is the value to be written into the calling processor's interrupt management area's current processor priority register (cppr).

Semantics:

- If the platform implements the Platform Reserved Interrupt Priority Level Option, and the priority field of the xirr parameter matches one of the reserved interrupt priorities then return H_Resource.
- The value of the cppr parameter is written into the calling processor's cppr causing the interrupt source controller to reject any interrupt of equal or less favored priority. Then hypervisor returns H_Success or H_Hardware (if an unrecoverable hardware error occurred).

14.5.4.7.3 H_IPI

int64 hcall (const uint64 H_IPI, uint64 server#, uint64 mfrr)

Parameters:

- server#: The server number gotten from the **`"ibm,ppc-interrupt-server#s"`** property associated with the processor and/or thread to be interrupted.
- mfrr: The priority value the inter-processor interrupt to be signaled.

Semantics:

- If the platform implements the Platform Reserved Interrupt Priority Level Option, and the priority field of the xirr parameter matches one of the reserved interrupt priorities then return H_Resource.
- If the value of the server# parameter specifies of one of the processors in the calling processor's partition, then the value in the low order byte of the mfrr parameter is written into the mfrr register (BA+12) of the processor's interrupt management area causing that interrupt source controller to signal an "inter-processor interrupt" (IPI) to the processor associated with the specified interrupt management area. Hypervisor then returns H_Success or H_Hardware (if an unrecoverable hardware error occurred). If the server# value is not legal, hypervisor returns H_Parameter.

  - If the Shared Logical Resource option is implemented and the server# parameter represents a shared logical resource location that has been rescinded by the owner, return H_RESCINDED.

14.5.4.7.4 H_IPOLL

int64 hcall (const uint64 H_IPOLL, uint64 server#)

Parameters:

- server#: The server number gotten from the **"ibm,ppc-interrupt-server#s"** property associated with the processor and/or tread to be interrupted.

Semantics:

- If the value of the server# parameter specifies of one of the processors in the calling processor's partition, then hypervisor reads the 4 byte contents of the processor's interrupt management area port at offset BA+0 into the low order 4 bytes of register R4 and the one byte of the mfrr (BA+12) into the low order byte of R5. Reading these addresses has no side effects and is used to poll for pending interrupts. Hypervisor then returns H_Success or H_Hardware (if an unrecoverable hardware error occurred). If the server# value is not legal, hypervisor returns H_Parameter.
  - If the Shared Logical Resource option is implemented and the server# parameter represents a shared logical resource location that has been rescinded by the owner, return H_RESCINDED.

14.5.4.7.5 H_XIRR / H_XIRR-X

These hcall()s provide the same base function that is they return the interrupt source number associated with the external interrupt. H_XIRR-X further supplies the time stamp of the interrupt . Legacy implementations implement only H_XIRR, returning H_Function for a call to H_XIRR-X. POWER8 implementations also implement H_XIRR-X.

int64 hcall (const uint64 H_XIRR)

```
int64                       /* H_Success: expected return code */
                            /* H_Hardware: The hcall() experienced a hardware fault potentially */
                            /* preventing the function */
    hcall (const uint64 H_XIRR-X,/* Accept an interrupt returning the external interrupt request register */
           uint8);
```

Parameters:

- H_XIRR: no input parameters defined.
- H_XIRR-X: cppr: the internal current processor priority of the calling virtual processor. Valid values in the range of 0x00 – most favored to 0xFF – least favored less those values specified by the **"ibm,plat-res-int-priorities"** property in the **root** node).

Semantics:

- Hypervisor reads the 4 byte contents of the processor's interrupt management area port at offset BA+4 into the low order 4 bytes of the register R4. Reading this address has the side effect of accepting the interrupt and raising the current processor priority to that of the accepted interrupt.
- Place the timestamp when the hypervisor first received the interrupt into R5.
- Hypervisor then returns H_Success or H_Hardware (if an unrecoverable hardware error occurred).

14.5.4.8 Memory Migration Support hcall()s

To assist an OS in memory migration, the following hcall() is provided. During the migration process, it is the responsibility of the OS to not change the DMA mappings referenced by the translations buffer (for example by using the H_GET_TCE, H_PUT_TCE hcall()s, or other DMA mapping hcall()s). Failure of the OS to serialize such DMA mapping access may result in undesirable DMA mappings within the caller's partition (but not outside of the caller's partition). Further, it is the responsibility of the OS to serialize calls to the H_MIGRATE_DMA service relative to the

logical bus numbers referenced. Failure of the OS to serialize relative to the logical bus numbers may result DMA data corruption within the caller's partition.

On certain implementations, DMA read operations targeting the old page may still be in process for some time after the H_MIGRATE_DMA call returns; this requires that the OS not reuse/modify the data within the old page until the worst case DMA read access time has expired. The **"ibm,dma-delay-time"** property (see Section B.6.3.1, "RTAS Node Properties," on page 690) gives the OS this implementation dependent delay value. Failure to observe this delay time may result in data corruption as seen by the caller's I/O adapter(s).

R1–14.5.4.8–1. For the LPAR option supporting the hcall-migrate function set: The platform must supply the **"ibm,dma-delay-time"** property under the **/rtas** node of the device tree.

Memory pages may be simultaneously mapped by multiple DMA agents, with different translation table formats and operation characteristics. The H_MIGRATE_DMA hcall() atomically performs the memory migration process so that the new page contains the old page contents (as updated by any DMA write operations allowed during migration), with all DMA mappings and engines directed to access the new page. The entries in the mapping list contain the logical bus number associated with the mapping and the I/O address of the mapping. From these two data, the hcall() associates the using DMA agent, that agent's DMA control procedures, the specific mapping table and mapping table entry.

R1–14.5.4.8–2. For the LPAR option supporting the hcall-migrate function set: The platform must support migration of pages mapped for DMA using any of the platform supported DMA agents.

R1–14.5.4.8–3. For the LPAR option supporting the hcall-migrate function set: All the platform's DMA agents must support mechanisms that enable the platform to meet the syntax, semantics and requirements set forth in section 14.5.4.8.1.

Implementation Note: The minimal hardware mechanisms to support the hcall-migrate function set are to quiesce DMA operation, flush outstanding data to their targets (both reads and writes), modify their DMA mapping and re-enable operation utilizing said modified DMA mapping without introducing unrecoverable operational failures. Provision for the hardware to direct DMA write operations to both old and new pages provides a significantly more robust implementation.

It is the intent of this architecture to have all memory in the platform have the capability to be migrated. However, on the rare implementation that cannot meet that intent, the **"ibm,no-h-migrate-dma"** property may be provided in **memory** nodes for which H_MIGRATE_DMA cannot be implemented.

R1–14.5.4.8–4. For the LPAR option supporting the hcall-migrate function set: If a memory node cannot support H_MIGRATE_DMA, then that **memory** node must contain the **"ibm,no-h-migrate-dma"** property.

For the I/O Super Page option the I/O page size is an attribute of the specified LIOBN (I/O pages mapped by a given LIOBN are a uniform size), also the syntax and semantics of H_MIGRATE_DMA are extended to allow migration of I/O pages that are larger than 4K bytes and have more than 256 xlates translation entries. Specifying more than 256 translation entries requires a sequence of calls to H_MIGRATE_DMA with the same "newpage" address. Making a call in the sequence with a length parameter of zero terminates the operation – should this termination happen after the start of the physical migration, the resulting state of the calling partition's memory is unpredictable. Failure to make a continuing call in the sequence for more than one second aborts the operation; again the resulting state of the calling partition's memory is unpredictable.

The introduction of super pages introduces the case where portions of the super page may be I/O mapped and thus require the use of H_MIGRATE_DMA to move the logical super page from one physical page to another even though the super page as a whole may not be I/O mapped. To handle this case, the LIOBN value of 0xFFFFFFFF is reserved to allow the specification, within an translations entry (passed to H_MIGRATE_DMA via the xlates parameter), of a super page that is not currently I/O mapped. In this case, the normally reserved byte at xlates entry offset 4 is used to specify the power of two size of the super page.

R1–14.5.4.8–5. For the I/O Super Page option: the platform must support the setting by the client of byte 3 bit 0 of the **`ibm,architecture.vec 5`** as input to the **`ibm,client-architecture-support`** method.

14.5.4.8.1 H_MIGRATE_DMA

```
int64                                   /*H_Success: Expected completion return code,
                                        H_Parameter: a parameter is invalid,
                                        H_LongBusyOrder1msec,
                                        H_LongBusyOrder10msec,
                                        H_Function
                                        For the Shared Logical Resource Option: H_ H_RESCINDED*/
                                        /* For the I/O Super Page Option, the following additional return
                                        codes are defined:
                                        H_CONTINUE: More translations are needed to complete the
                                                request
                                        H_P3: The length parameter did not contain the next expected
                                                value in the call sequence.
                                        H_Resource: Insufficient resources to perform the request
                                        H_MEM_PARM: The first xlate entry specifying the LIOBN of
                                         0xFFFFFFFF contains an unsupported page size specification
                                         or invalid logical real address. */
    hcall (const uint64 H_MIGRATE_DMA,  /*Migrates a page mapped by one or more DMA mappings*/
    uint64 newpage,                     /*Logical address of new DMA target page*/
    uint64 xlates,                      /*List of translations to current DMA target page*/
    uint64 length);                     /*Length of translation list*/
```

Parameters:

- newpage (The Logical address of the new page to be the target of the TCE translations)
- xlates (The Logical address of a list of translations against the target page the format of this list is:
  - List starts on a page (4 K) boundary.
  - Contains up to 256 translation entries:
    - First 4 bytes of a translation entry is the logical bus number as from either the:
      - **`"ibm,dma-window"`** property
      - or the reserved LIOBN 0xFFFFFFFF.
    - Next 12 bytes of a translation entry is the logical bus offset (I/O bus address). The format of the I/O bus address is dependent upon the DMA agent:
      - For 32 bit PCI, the high order 8 bytes are reserved with the low order 4 bytes containing a 4 K aligned address (low order 12 bits =zero).
      - For 64 bit PCI, the high order 4 bytes are reserved with the low order 8bytes containing a 4 K aligned address (low order 12 bits =zero).
      - For the I/O Super Page option the very first translation entry passed is for the largest I/O page to be migrated by this sequence of calls; else all translation entries are for the single 4K byte logical page being migrated. The first translation entry may either be a current I/O mapping for the largest I/O page that the caller wishes to migrate, or the first translation entry may use the reserved LIOBN number of 0xFFFFFFFF, with the next byte indicating the page size as 2**N where N is the numeric value of the byte at offset 4 into the translation

entry with the low order 8 bytes of the translation entry being the logical real address of the start of the page to be migrated (the low order N bits = zero).

- length (Number of entries in translation list is less than or equal to 256)
  - If the total number of translation entries in the xlates list is less than or equal to 256 then the "length" parameter is the number of translation entries.
  - For the I/O Super Page option and specifying more than 256 translation entries, the client makes a series of calls, each passing 256 translation entries with the "length" parameter being the negative of the total number of translation entries yet to be passed until there are less than or equal to 256 remaining then for the final call in the initiating sequence the "length" parameter is positive as above.

Semantics:

- For the I/O Super Page option: determine if a migration operation is in process for this "newpage" address:
  - Then:
    - If the previous hcall() for the migration operation was more than 1 second ago, return H_Aborted.
    - If the length parameter value is zero then abort the migration operation and return H_TERM.
    - If the length parameter value is not the next expected in the sequence return H_P3.
    - Record the new xlates
    - If the length parameter is less than zero return H_CONTINUE.
  - Else
    - If the number of outstanding operations is more than an implementation specific number as communicated in the **"ibm,vec-5"** property then return H_Resource
    - If the length parameter is less than zero, initiate a new migration operation for the "newpage" address. (Note resources for the operation may be allocated at this point and freed when the operation terminates either normally, in error, or via timeout. Implementations may, in unusual cases, use a busy return code to wait for the release of resources from an immanently completing operation.
    - The first xlate entry specifies the length and starting address of the page to be migrated, if this specification is invalid (unsupported length, the address is invalid for the partition, or not aligned to the length) return H_MEM_PARM.
    - If the operation specifies more than an implementation specific number of xlates as communicated in the **"ibm,vec-5"** property then return H_Resource.
- Check that the page to be migrated can be migrated, else H_PARAMETER.
- Check that the newpage is within the allocated logical page range of the calling partition and the address is aligned to the I/O page size of the first translation entry passed else H_PARAMETER.
  - If the Shared Logical Resource option is implemented and the newpage parameter represents a shared logical resource location that has been rescinded by the owner, return H_RESCINDED.
- The contents of the xlates buffer are checked.
  - This may be done as each entry is used, or it may be done prior to starting the operation.
    - If the former, then partial processing must be backed out in the case of a detected parameter error.

- ❑ If the later, then the translation entries must be copied into an area that is not accessible by the calling OS to prevent parameter corruption after they have been verified. The OS perceived reentrancy of the function is not diminished if this option is chosen.

- The xlates buffer starts on a 4 K boundary within the partition's logical address range else H_PARAMETER.
- The length parameter is between (for the I/O Super Page option: the negative of the maximum number of xlate entries supported as indicated in the **"ibm,architecture-vec-5"** property of the **/chosen** device tree node else 1) and 256 else H_PARAMETER.
- For the I/O Super Page option: the length of the physical page to be migrated is the length of the I/O page of the first translation entry; else the length of the physical page to be migrated is 4K bytes.
- Each translation originally references the same physical page, or a portion there of, else H_PARAMETER.
- Each logical bus offset is within the allocated range of the calling partition else H_PARAMETER.
  - ❑ If the Shared Logical Resource option is implemented and the logical bus offset represents a shared logical resource location that has been rescinded by the owner, return H_RESCINDED.
- Check the logical bus number:
  - ❑ Is allocated to the calling partition else H_PARAMETER.
    Or: If the Shared Logical Resource option is implemented and the logical bus number represents a shared logical resource location that has been rescinded by the owner, return H_RESCINDED.
  - ❑ For the I/O Super Page option: if the LIOBN implies a larger page size than that specified by the first translation entry for this migrate operation, place the index of the translation entry (0-255) into register R4 and return H_PGSB_PARM.
  - ❑ If the LIOBN referenced an unsupported DMA agent, place the index of the translation entry (0-255) into register R4 and return H_Function.
  - ❑ If the logical bus number is not supported, return H_PARAMETER.

Note: The following is written from the perspective of a PCI DMA agent; other DMA agents may require a different sequence of operations to achieve equivalent results.

- The hypervisor disables arbitration for the IOA(s) associated with the translation entries. (In some cases, where multiple IOAs share a given TCE range, arbitration must be disabled for multiple IOAs. The firmware assigned the bus address ranges to each IOA so knows which IOAs correspond to which translation.)
- Waits for outstanding DMA write activity to complete. (This is accomplished by doing a load from an appropriate register the bridge(s) closest to the IOA -- when the load completes (dependency on load data is satisfied) all DMA write activity has completed.)
- The hypervisor copies the contents of the 4 K page originally accessed by the TCE(s) to the page referenced by the newpage value.
- The hypervisor translates the logical address within the newpage parameter and stores the resultant value in the TCE table entries specified by the translation entries.
- Executes a sync operation to ensure that the new TCE data is visible.
- The hypervisor enables arbitration on the IOA(s) associated with the translation entities and returns H_Success.

Implementation Notes:

1. The firmware should be written to minimize the arbitration disable time. The old page should be read into cache (possibly using the data cache touch operations) prior to disabling the arbitration. Implementation dependent algorithms can significantly improve the page copy time.

2. The firmware does not have to serialize this hcall() with other hcall()s as long as it updates the TCE using atomic eight (8) byte write operations. However, if the OS does not serialize this call with H_PUT_TCE to the same TCE, and with other H_MIGRATE_DMA calls to the same IOA(s) the calling LPARs DMA buffers could be corrupted.

3. To minimize the effect of such unsupported DMA agents, the platform designer should isolate such agents on their own bus with their own **"ibm,dma-window"** property specification.

14.5.4.9 Performance Monitor Support hcall()s

14.5.4.9.1 H_PERFMON

To manage the Performance Monitor Function:

```
int64                          /* H_Success,
                               H_HARDWARE,  Hardware error
                               H_PARAMETER, Unsupported mode bit
                               H_BUSY,                Try again
                               H_RESOURCE   Conflicting resources in use*/
    hcall (const uint64 H_PERFMON,          /* Function code */
    uint64 mode-set,                /* Platform Modes to enable */
    uint64 mode-reset);             /* Platform Modes to reset */
```

Parameters:

- mode-setPlatform specific modes to be set by this call
- mode-resetPlatform specific modes to be reset by this call

Semantics:

- mode-set bit(s) check for platform specific validity else H_PARAMETER
- mode-reset bit(s) check for platform specific validity else H_PARAMETER
- if any mode-set bits are set, activate corresponding mode(s) - if logically capable else H_RESOURCE
- if any mode-reset bits are on, deactivate corresponding mode(s) - if logically capable else H_RESOURCE
- place current state of platform specific modes in R4, return H_Success

Defined Perfmon mode bits:

bit 0: 1= Enable Perfmon

bit1: 0= Low threshold granularity 1= High threshold granularity

14.5.4.10 H_GET_DMA_XLATES_LIMITED

This hcall returns the I/O bus address of the first entry defined for the specified LIOBN and the corresponding logical address within the range beginning with the Start logical address and less than the End logical addresses, the search is limited to the range of I/O bus addresses specified by the SIOBA and EIOBA parameters.

R1–14.5.4.10–1. For the LRDR Option: The platform must implement the H_GET_DMA_XLATES_LIMITED hcall() per the syntax and semantics specified in section14.5.4.10, “H_GET_DMA_XLATES_LIMITED,” on page 436.

R1–14.5.4.10–2. For the LRDR Option: The platform must present the **`"ibm,h-get-dma-xlates-limited-supported"`** property in all PCI host bridge OpenFirmware nodes for which the H_GET_DMA_XLATES_LIMITED hcall() is supported for all child LIOBNs.

Syntax:

```
int64                   /*H_Success: Expected return code */
                        /*H_PARAMETER: Invalid logical I/O bus number specified*/
                        /*H_P2: Invalid starting logical address */
                        /*H_P3: Invalid ending logical address*/
                        /*H_P4: Invalid start I/O Bus Address*/
                        /*H_P5: Invalid end I/O Bus Address*/
                        /*H_IN_PROGRESS: Call is in progress, end of table was not reached*/
                        /*H_PARTIAL: Partial completion*/
                        /*H_PAGE_REGISTERED: Page match and last page of table*/
  hcall (const uint64 H_GET_DMA_XLATES_LIMITED, /*Return I/O Bus and corresponding logical address*/
        uint32 LIOBN,           /*Logical I/O Bus Number of a translation table*/
        uint64 SLA,             /*Starting logical address of a range*/
        uint64 ELA,             /*Ending logical address of a range*/
        uint64 SIOBA,           /*Start I/O Bus Address*/
        uint64 EIOBA);          /*End I/O Bus Address */
```

Parameters:

- Register R4: Logical I/O Bus Number (LIOBN)
  - Bits 0-31are reserved and set to zero.
  - Bits 32-63 contain a 32-bit unsigned binary integer that identifies a translation which may have one or more entries that translate to a page within a range specified by the Start and End logical addresses.
- Register R5: Start Logical Address (SLA)
- Register R6: End Logical Address (ELA)
- Register R7: Start I/O Bus Address (SIOBA) of the translation specified by the LIOBN
  - The SIOBA register may specify a special value of -1 or a starting IOBA
- Register R8: End I/O Bus Address (EIOBA) of the translation specified by the LIOBN
  - The EIOBA register may specify a special value of -1 or an ending IOBA

Semantics:

- Check that the specified LIOBN is supported and allocated to the calling logical partition, else H_PARAMETER.
- Check that the specified start logical address (SLA) is within the allocated range of the calling logical partition, and is designated on a 4 K-byte boundary, else H_P2.
- Check that the End logical address (ELA) minus 4K is within the allocated range of the calling logical partition, and is designated on a 4 K-byte boundary, else H_P3. (May point no further than one page beyond the maximum partition logical real address in order to stay within the partition yet include the last partition page in the range of the test.)

- Check that the specified starting logical address (SLA) is less than the specified ending logical address (ELA), else H_P2.
- Check that the page specified by the logical addresses within the specified range is within the allocated range of the calling logical partition and the address is 4 K-byte aligned else H_P2.
- Check the content of SIOBA
  - If a value other than -1 is specified, check that the specified start I/O bus address (SIOBA) is not outside of the range of IOBAs for the specified LIOBN, else H_P4.
  - If the SIOBA specifies a value of -1, the hypervisor starts the search at the lowest IOBA in the translation table, otherwise the search starts at the address specified by the SIOBA.
- Check the content of EIOBA
  - If a value other than -1 is specified, check that the specified ending I/O bus address (EIOBA) is not outside of the range of IOBAs for the specified LIOBN, else H_P5.
  - If the EIOBA specifies a value of -1, the hypervisor ends the search at the highest IOBA in the translation table, otherwise the search ends at the address specified by the EIOBA.

Outputs:

Place the I/O bus address and corresponding logical address into the respective registers:

- Register R4: I/O Bus Address (IOBA)
  - This register contains a 64-bit unsigned binary integer that specifies the I/O bus address of the page within the specified logical address range for the specified LIOBN.
  - The IOBA is returned when the hcall() completes with either H_PARTIAL, H_PAGE_REGISTERED, or H_IN_PROGRESS return codes.
- Register R5: Corresponding Logical Address (CLA)
  - This register contains a 64-bit unsigned binary integer that designates the logical address of a page within the specified range that corresponds to the I/O bus address.
  - If the hcall() completes with H_IN_PROGRESS return code, the corresponding logical address (CLA) is not returned.
- When the hcall() completes with H_PARTIAL or H_PAGE_REGISTERED return code:
  - The I/O bus address (IOBA) and corresponding logical address (CLA) are returned.
- When the hcall() completes with H_PAGE_REGISTERED return code:
  - The I/O bus address (IOBA) is for the final page of the translation table for the specified LIOBN as limited by the EIOBA parameter.
- When the hcall() completes with H_IN_PROGRESS return code:
  - The current IOBA being searched against the specified range is returned, but the corresponding logical address is not returned.
  - The hcall can be reissued by specifying the IOBA as the starting IOBA without incrementing the IOBA by the resource page size.

Firmware Implementation Notes:

1. When the H_GET_DMA_XLATES_LIMITED hcall() is issued, the hypervisor searches the translation table designated by the specified LIOBN, from the entry for SIOBA through the entry for EIOBA in IOBA order, for the entries that translate to a page within a given range of logical addresses. If an entry is found, the hcall() completes with the H_PAGE_REGISTERED return code if the page found is the last entry in the translation table, or the H_PARTIAL return code for all other pages, and the IOBA with the corresponding logical address are returned in output registers R4 and R5 respectively.

2. The hypervisor searches the translation table in IOBA order, and proceeds in that order until an entry that translates to a physical address within the specified range of logical addresses is found, in which case, the hcall() completes with H_PARTIAL or H_PAGE_REGISTERED return code, or H_Success, if the end of the translation table, as specified by the EIOBA parameter, is reached.

Software Implementation Notes:

1. When the hcall() completes with H_PARTIAL return code, the stored IOBA is incremented by the page size of the resource corresponding to the specified LIOBN, and then specified as the starting I/O bus address on a subsequent call where the hypervisor would then proceed with the search until the end of the translation table, specified by the EIOBA parameter, is reached. The caller can accumulate a full list of the IOBAs for the specified LIOBN that translate into the specified range of logical addresses, which then forms part of the xlate translation entries specified as an input to the H_MIGRATE_DMA function.

2. When the hcall() completes with H_PAGE_REGISTERED return code, this indicates that page is contained in the specified range of logical addresses, and it is the last page of the translation table such that the search for that LIOBN is complete.

3. If a value other than -1 is specified in the starting I/O bus address register, the program should check that the specified SIOBA value is not the same as the returned IOBA.

14.6 RTAS Requirements

RTAS function as specified in this architecture is still required for LoPAPR LPAR partition. RTAS is instantiated via an OF client interface call. RTAS operates without memory translation, therefore, the OS should instantiate it within the RMA, however, the OF client interface does not enforce this limitation. The RTAS calling sequences remain unchanged. However, in LPAR configurations RTAS code is implemented differently than in non-LPAR systems. LPAR RTAS has a part which is replicated in each partition, and since RTAS has the capability to manipulate hardware system resources, RTAS has a part which is implemented in the hypervisor. In the hypervisor, there is a check of the RTAS parameters for validity before execution. Therefore, the function of the partition replicated RTAS call is to martial the arguments and make the required hidden hcall()s to the hypervisor. In a non-LPAR system, RTAS calls are assumed to be made with valid parameters. This cannot be assumed with LPAR. The LPAR RTAS operates by all the rules of non-LPAR RTAS relative to it running real, with real mode pointers to arguments and the same serialization requirement relative to a single partition. However, the hypervisor may not assume that the caller is following these serialization rules, failure on the part of the OS to properly serialize is allowed to cause unpredictable results within the scope of the calling partition but may not affect the proper operation of other platform partitions.

The following is a list of RTAS functions that are not defined or implemented when the LPAR option is active:

- *restart-rtas*

R1–14.6–1. For the LPAR option: The platform must implement the PowerPC External Interrupt option.

R1–14.6–2. For the LPAR option: The Firmware must initialize each processor's interrupt management area's CPPR to the most favored level and its MFRR to the least favored level before passing control of the processor to the OS.

R1–14.6–3. For the LPAR option: The RTAS rules of serialization of RTAS calls must only apply to a partition and not to the system.

R1–14.6–4. For the LPAR option: The hypervisor cannot trust the RTAS calls to have no errors, therefore, the hypervisor must check a partition's RTAS call parameters for validity.

R1–14.6–5. For the LPAR option: RTAS must be instantiated within the RMA of partition storage.

R1–14.6–6. For the LPAR option: RTAS arguments must be within the RMA of partition storage unless specifically specified in the RTAS call definition.

R1–14.6–7. For the LPAR option: If one or more hcalls fail due to hardware error (return status -1), the platform must make available, prior to the completion of the next boot sequence, via an *event-scan/check-exception*, an error log indicating the hardware FRU responsible for such failures. Due to the asynchronous nature of error analysis, there is not a direct correlation between the log and a specific failing hcall(), indeed the error log may precede the failing hcall().

14.7 OF Requirements

The hypervisor is initialized and configured prior to the loading of OF into the partition and boot of any client program (OS) in the partition by OF. The NVRAM data base that describes the platform's partitioning is used to trigger the loading and initialization of the hypervisor. When Logical Partitioning is enabled, a copy of OF code is loaded into each partition where it builds the per partition device tree within the partition's RAM. The per partition device tree contains only entries for platform components actually assigned to or used by the partition. The invocation of the subset of the OF Client interface specified below appears the same to the OS image regardless of the state of the LPAR option.

A model of the boot sequence is as follows:

1. Support processor runs chip tests and configures the CPU chips.
2. The support processor loads the boot ROM image into System Memory along with the configuration information.
    a. POST code
    b. Initialization Firmware
    c. Hardware configuration reporting structures
    d. OF
    e. Hypervisor RTAS
3. boot ROM executes POST and Initialization Firmware.
4. Processor initialization code synchronizes the time bases of all platform processors to a small value (approaching zero).
5. Initialization Firmware accesses the NVRAM Partition Database to determine if the LPAR option is enabled.
6. Initialization Firmware initializes the hypervisor.
7. The hypervisor configures itself using the hardware configuration reporting structures.
8. The hypervisor configures the various partitions with resources as required by the NVRAM Partition Database.
9. The hypervisor loads a copy of OF into each partition passing to OF a resource reporting structure known as the NACA/PACA.

10. OF notices in the NACA/PACA that a specific partition table is specified.
11. OF Scans the configuration and walks the buses to build the partition device tree.
12. OF requests the specific partition table from the NVRAM Partition Database.
13. OF loads RTAS into the partition's memory.
14. OF pulls in the configuration variables from the partition's NVRAM area and uses them to determine the partition's boot device.
15. OF then loads the client program and starts executing it with one of the partition's processors.
16. The client program notices that it is running on a LPAR capable machine but does not have the hypervisor bit on in the MSR so must use hcall() routines for its PFT and TCE accesses. The presence of the **"ibm,hypertas-functions"** property is a duplicate indication of LPAR mode.

R1–14.7–1. For the LPAR option: The OF code state must be retained after all partitions are initialized pending future boot requests.

R1–14.7–2. For the LPAR option: The OF code must recognize that logical partitioning is required as opposed to a non-LPARed system.

R1–14.7–3. For the LPAR option: The OF must generate the device tree for the partition within the partition's RAM.

R1–14.7–4. For the LPAR combined with Dynamic Reconfiguration option: The **"interrupt-ranges"** property for any reported interrupt source controller must report all possible interrupt source numbers.

R1–14.7–5. For the LPAR option: The OF device tree for a partition must include in the root node, the **"ibm,partition-no"** property.

R1–14.7–6. For the LPAR option: The OF device tree for an LPAR capable model not running in a partition must include in the root node, the **"ibm,partition-no"** property when the default partition number for the first partition created is not 1.

R1–14.7–7. For the LPAR option: The **"ibm,partition-no"** property value must be an integer in the range of 1 to 2^{20}-1.

R1–14.7–8. For the LPAR option: The OF device tree for a partition must include in the root node, the **"ibm,partition-name"** property.

R1–14.7–9. For the LPAR option: When the platform does not provide a partition manager and the one and only partition in the system owns all the partition visible system resources, then the default value of the **"ibm,partition-name"** property must be the content of the SE keyword (as displayed in the same form as the root node **"system-id"** property) with a hyphen added between the plant of manufacture and sequence number.

R1–14.7–10. For the LPAR option: The nodes of the OF device tree for a partition that represent platform resources that are not explicitly allocated for the control of the platform's OS image must be marked "used-by-rtas". This includes, but is not limited to, memory controllers, and IO bridges that are a part of the platform's infrastructure common to more than one partition and commonly represented in the OF device tree. But does not include read only resources such as environmental sensors.

R1–14.7–11. For the LPAR option: The OF must, at the OS's request, load the required RTAS into the partition's real addressable memory region.

R1–14.7–12. For the LPAR option: The OF must use the partition's segment of the NVRAM to establish the partition's boot device and configuration variables.

R1–14.7–13. For the LPAR option: The OF must load the client program and choose the partition's processor on which to begin execution.

Note: It is the responsibility of the client program to recognize whether or not to use LPAR page management.

R1–14.7–14. For the LPAR option: The platform must initialize the time base of the first processor to a small (approaching zero) value prior to turning over control of the processor to a client program.

R1–14.7–15. *(Merged into Requirement R1–11.1–9)*

R1–14.7–16. For the LPAR option: The OF Client Interface must restrict access to only resources contained within the calling partition's version of the device tree.

R1–14.7–17. For the LPAR option: The OF Client Interface must prevent the calls of one partition's client program from interfering with the operation of another partition's client program.

R1–14.7–18. For the LPAR option: The OF Client Interface must restrict its supported calls and methods to those specified in Table 181, "OF Client Interface Functions Supported under the LPAR Option," on page 442.

R1–14.7–19. For the LPAR option: Any hidden hcall()s which firmware may use to implement OF functions must check its parameters to insure compliance with all of the architecturally mandated OF requirements.

R1–14.7–20. For the LPAR option: The OF Client Interface functions "start-cpu" and "resume-cpu" must restrict their operation to processors assigned to the calling Client's partition.

Table 181. OF Client Interface Functions Supported under the LPAR Option

test	cannon	child	finddevice
getprop	getproplen	instance-to-package	instance-to-path
nextprop	package-to-path	parent	peer
setprop	call-method	test-method	close
open	read	seek	write
claim	release	boot	enter
exit	start-cpu	milliseconds	size(/chosen/nvram)
get-time		instantiate-rtas	

14.8 NVRAM Requirements

The NVRAM is divided into multiple partitions each containing different categories of data similar to files in a file system (these NVRAM partitions are not to be confused with LPAR partitions). Each NVRAM partition is structured with a self identifying header followed by its partition unique data. Many of these NVRAM partitions contain data only relevant to the platform firmware, while others contain data that either is for OS image use from boot to boot or is used to communicate operational parameters from the OS image to the platform. The platform firmware on LPAR supporting platforms structures the NVRAM as per Table 182, "LPAR NVRAM Map," on page 443. Each LPAR partition is assigned a region of NVRAM space. This includes space for LPAR partition specific configuration variables as well as the minimum 4 K space reserved for the OS image. The hypervisor restricts access for the LPAR partition, through logical address translation and range checking, to its assigned NVRAM region. Other regions of NVRAM are reserved for firmware use including, for instance, information about how the system should be partitioned.

Table 182. LPAR NVRAM Map

Real Address Range	Per Partition NVRAM access routine rtas call Logical Address Range -- outside of legal range return 0x00 and discard write data.	Contents
0x00 to F-1	NA	Firmware only partitions (Signatures 0x00 to 0x6F)
F to (F-1+P)	0x00 to P	Per LPAR partition copies of supported NVRAM partitions with signatures 0x70 to 0x7F
(F+P) to (F-1+2P)	0x00 to P	Per LPAR partition copies of supported NVRAM partitions with signatures 0x70 to 0x7F
. . .		
(F+(P*(n-1))) to ((F-1)+ nP)	0x00 to P	Per LPAR partition copies of supported NVRAM partitions with signatures 0x70 to 0x7F

Table 183. NVRAM partitions on LPAR platforms

Visible to:	Partition Signatures	Partition Name	Comments
Only to the Platform firmware	0x00 - 0x6F		
Only to Platform firmware and the OS image running in the owning LPAR Partition. The read and write NVRAM RTAS routines	0x70	Common	This partition is duplicated per partition.
	0x7F	0x77777777777777777777777	This partition is duplicated per partition and is at least 4 KB long when the OS is first installed.

R1–14.8–1. For the LPAR option: Platform OF must locate configuration variables that the OS must manipulate to select options as to how the specific OS image interfaces or relates to the platform in the partition's "system" partition signature (0x70) named "common", specifically none may be located in the "OF" signature (0x50).

R1–14.8–2. For the LPAR option: The NVRAM region assigned to an LPAR partition must contain, after any platform required NVRAM partitions have been allocated, a free space partition a minimum of 4 KB long prior to the installation of the partition's OS image.

14.9 Administrative Application Communication Requirements

The platform needs to communicate with the an administrative application (outside of the scope of LoPAPR) to manage the platform resources. The administrative application may run in an external computer such as a Hardware Management Console, or it may be integrated into a service partition. Many system facilities are not dedicated to an LPAR partition but are managed through the HMC and the administrative application.

R1–14.9–1. For the LPAR option: The platform must provide a communications means to the administrative application.

R1–14.9–2. For the LPAR option: The platform must respond to messages received from the administrative application.

14.10 RTAS Access to Hypervisor Virtualized Resources

All allolcatable platform resources are always assigned to a partition. There always exists a dummy partition that is never active. Resources assigned to partitions that are inactive may be reassigned to other partitions by mechanisms implemented in the Hardware Management Console.

R1–14.10–1. For the LPAR option: The *nvram-fetch* RTAS call must restricted access to only the LPAR partition's assigned "OS", "System" and "Error Log" nvram partitions.

R1–14.10–2. For the LPAR option: The *nvram-store* RTAS call must restricted accss to only the LPAR partition's assigned "OS", "System" and "Error Log" nvram partitions.

R1–14.10–3. For the LPAR option: The *get-time-of-day* RTAS call must return the LPAR partition's specific time of day clock value.

R1–14.10–4. For the LPAR option: The *set-time-of-day* RTAS call must set the LPAR partition's specific time of day clock value.

Firmware Implementation Note: The model implementation keeps time of day on a partition basis. What is really changed is the offset from the hardware TOD clock which is not normally written (Only written if for some reason it is approaching its maximum value, such as after a battery failure).

R1–14.10–5. For the LPAR option: The *event-scan* RTAS call must report global events to each LPAR partition and LPAR partition local events only to the affected LPAR partition.

R1–14.10–6. For the LPAR option: The *check-exception* RTAS call must report global events to each LPAR partition and LPAR partition local events only to the affected LPAR partition.

R1–14.10–7. For the LPAR option: The *rtas-last-error* RTAS call must report only RTAS errors affecting the calling LPAR partition.

R1–14.10–8. For the LPAR option: The *ibm,read-pci-config* RTAS calls must restrict access to only IOAs assigned to the calling LPAR partition, and if the configuration address is not available to the caller, must return a status of Success with all ones as the output value.

R1–14.10–9. For the LPAR option: The *ibm,write-pci-config* RTAS calls must restrict access to only IOAs assigned to the calling LPAR partition, and if the configuration address is not available to the caller, must be ignored and must return a status of Success.

R1–14.10–10. For the LPAR option: The *ibm,write-pci-config* RTAS calls must prevent changing of the firmware assigned interrupt message number on IOAs configured to use message signaled interrupts.

R1–14.10–11. For the LPAR option: The platform must virtualize the *display-character* RTAS call such that the operator can distinguish and selectively read messages from each partition without interference with messages from other partitions.

R1–14.10–12. For the LPAR option: The *set-indicator* RTAS call must restrict access to only indicators assigned to the calling LPAR partition.

R1–14.10–13. For the LPAR option: The effects of the *system-reboot* RTAS call must be restricted to only the calling LPAR partition.

Firmware Implementation Note: One standard OS response to a machine check is to reboot. Thus expecting the firmware to reset any error conditions such as in the I/O sub-system. When the I/O sub-system, or parts thereof, are shared among multiple partitions, the platform cannot allow the boot of one partition to prevent another partition from detecting that it was also affected by an I/O error.

R1–14.10–14. For the LPAR option: The platform must deliver machine check and other event notifications to all affected partitions before initiating recovery operations such as rebooting and resetting hardware fault isolation circuits.

R1–14.10–15. For the LPAR option: The *start-cpu* RTAS call must be restricted to only the processors assigned to the calling LPAR partition.

R1–14.10–16. For the LPAR option: The *query-cpu-stopped-state* RTAS call must be restricted to only the processors assigned to the calling LPAR partition.

R1–14.10–17. For the LPAR option: The *power-off* and *ibm,power-off-ups* RTAS calls must deactivate the calling partition and not power off the platform if other partitions remain active.

R1–14.10–18. For the LPAR option: The *set-time-for-power-on* RTAS call must activate the platform when the partition requesting the earliest activation time is to be activated.

R1–14.10–19. For the LPAR option: The *ibm,os-term* RTAS call must adjust support processor surveillance to account for the termination of the LPAR partition's OS.

R1–14.10–20. For the LPAR option: The *ibm,set-xive* RTAS call must restrict access to only interrupt sources assigned to the calling LPAR partition by silently failing if the interrupt source is not owned by the calling partition (return success without modifying the state of the unowed interrupt logic).

R1–14.10–21. For the LPAR option: The *ibm,set-xive* RTAS call must restrict the written queue values to only interrupt processors assigned to the calling LPAR partition.

R1–14.10–22. For the LPAR option: The *ibm,get-xive* RTAS call must restrict access to only interrupt sources assigned to the calling LPAR partition by silently failing if the interrupt source is not owned by the calling partition (return success with the least favored interrupt level, the interrupt server number is undefined -- possibly all ones).

R1–14.10–23. For the LPAR option: The *ibm,int-on* RTAS call must restrict access to only interrupt sources assigned to the calling LPAR partition by silently failing if the interrupt source is not owned by the calling partition (return success without modifying the state of the unowed interrupt logic).

R1–14.10–24. For the LPAR option: The *ibm,int-off* RTAS call must restrict access to only interrupt sources assigned to the calling LPAR partition by silently failing if the interrupt source is not owned by the calling partition (return success without modifying the state of the unowed interrupt logic).

R1–14.10–25. For the LPAR option: The *ibm,configure-connector* RTAS call must restrict access to only Dynamic Reconfiguration Connectors assigned to the calling LPAR partition.

R1–14.10–26. For the LPAR option: The platform must either define or virtualize the power domains used by the *set-power-level* RTAS call such that power level settings do not affect other partitions.

R1–14.10–27. For the LPAR option: The *set-power-level* and *get-power-level* RTAS calls must restrict access to only power domains assigned to the calling partition.

R1–14.10–28. For the LPAR option: The platform must restrict the availability of the *ibm,exti2c* RTAS call to at most one partition (like any IOA slot).

R1–14.10–29. For the LPAR option: The *ibm,set-eeh-option* RTAS call must restrict access to only IOAs assigned to the calling partition.

R1–14.10–30. For the LPAR option: The *ibm,set-slot-reset* RTAS call must restrict access to only IOAs assigned to the calling partition.

R1–14.10–31. For the LPAR option: The *ibm,read-slot-reset-state2* RTAS call must restrict access to only IOAs assigned to the calling partition.

R1–14.10–32. For the LPAR option: The *ibm,configure-bridge* RTAS call must restrict access to only configuration addresses assigned to the calling partition.

R1–14.10–33. For the LPAR option: The *ibm,set-eeh-option* RTAS call must restrict access to only IOAs assigned to the calling partition.

R1–14.10–34. For the LPAR option: The platform must restrict the *ibm,open-errinjct, ibm,close-errinjct,* and *ibm,errinjct* RTAS calls as well as the errinjct properties be available on at most one partition as defined by a platform wide firmware configuration variable.

R1–14.10–35. For the LPAR option: Any hidden hcall()s which firmware may use to implement RTAS functions must check its parameters to insure compliance with all of the architecturally mandated RTAS requirements.

14.11 Shared Processor LPAR Option

The Shared Processor LPAR (SPLPAR) option allows the hypervisor to generate multiple virtual processors by time slicing a single physical processor. These multiple virtual processors may be assigned to one or more OS images. There are two primary customer advantages to SPLPAR over the standard LPAR. Most obviously, the assigned processing capacity of the partition can scale downwards to allow for more OS images to be supported on a single platform. The second customer advantage is that a SPLPAR platform can achieve higher processor utilization by providing partitions, that can use extra processing capacity, with the spare capacity ceded from other partitions. This allows the customer to take advantage of the variable nature of the instantaneous load on any one OS image to achieve an increase in the average utilization of the platform's capacity. While the peak capacity (directly related to the platform cost) stays constant, the customer may see a significant improvement in the average capacity among all the platform's workloads. However, since the peak capacity cannot be physically exceeded, the customer may experience a wider variance in performance when exercising the SPLPAR option.

In principal, the OS images running on the virtual processors of an SPLPAR platform need not be aware that they are sharing their physical processors, however, in practice, they experience significantly better performance if they make a few optimizations. Specifically, if the OS images cedes their virtual processor to the platform when they are idle, and confers their processor to the holder of a spin lock for which the virtual processor must wait. Another significant change due to SPLPAR is that there may not be a fixed relationship between a virtual processor and the physical processor that actualizes it. In those cases, such physical information as location codes are undefined, affinity and associativity values are indistinguishable, relationships to secondary caches are meaningless, and any attempt by an OS to characterize the quality of its processor (such as running diagnostics or performance comparisons to other virtual processors) provide unreliable results. OF entities, that represent physical characteristics of a virtual processor that do not remain fixed, take on altered definitions/ requirements in an SPLPAR environment.

To provide input to the capacity planning and quality of service tools, the hypervisor reports to an OS certain statistics, these include the minimum processor capacity that the OS can expect (the OS may cede any unused capacity back to the platform), the maximum processor capacity that the platform grants to the OS, the portion of spare capacity (up to the maximum) that the platform grants to the OS, and the maximum latency to a dispatch via an hcall().

The OS image optionally registers a data area (VPA) for each virtual processor using the H_REGISTER_VPA hcall(). The hypervisor maintains a variable, within the data area, that is incremented each time the virtual processor is dispatched/preempted, such that the dispatch variable is always even when the virtual processor is dispatched and always odd when it is not dispatched. The achitectural intent for the usage of the dispatch count variable is describe below in

the paragraph devoted to conferring the processor. Additionally this hcall() may register a trace buffer which the OS may activate to gain detailed information about virtual processor preemption and dispatching.

Both the VPA and the trace log buffer contain statistics on how long the virtual processor has waited (not been dispatched on a physical processor). Architecturally, the virtual processor wait time is divided into three intervals:

1. The time that the virtual processor waited to become logically ready to run again, for example:
    a. The time needed to resolve a fault
    b. The time needed to process a hypervisor preemption
    c. The time until a wake up event after voluntarily relinquishing the physical processor
2. The time spent waiting after interval 1 until virtual processor capacity was available. Shared processor partitions are granted a quantum of virtual processor capacity (execution time) each dispatch wheel rotation; thus if the partition has used its capacity, the ready to run virtual processor has to wait until the next quantum is granted.
3. The time spent waiting after interval 2 until the virtual processor was dispatched on a physical processor. This is arises from the fact that multiple ready to run virtual processors with virtual processor capacity may be competing for a single physical processor.

Two other performance statistics are available via hcall()s these are the Processor Utilization Register, and Pool Idle Count returned by the H_PURR and H_PIC hcall()s respectively. These two statistics are counts in the same units as counted by the processor time base. Like the time base, the PUR and PIC are 64 bit values that are set to a numerically low value during system initialization. The difference between their values at the end and beginning of monitored operations provides data on virtual processor performance. The value of the PUR is a count of processor cycles used by the calling virtual processor. The PUR count is intended to provide an indication to the partition software of the computation load supported by the virtual processor. SPLPAR virtual processors are created by dispatching the virtual processor's architectural state on one of the physical processors from a pool of physical processors. The value of the PIC is the summation of the physical processor pool idle cycles, that is the number of time base counts when the pool could not dispatch a virtual processor. The PIC count is intended to provide an indication to platform management software of the pool capacity to perform more work.

A well behaved OS image normally cedes its virtual processor to the platform using the H_CEDE hcall() after it determines that it currently has run out of useful work. The H_CEDE hcall() gives up the virtual processor until either an external interrupt (including decrementer, and Inter Processor Interrupt) or another one of the partition's processors executes an H_PROD hcall() see below. Note the decrementer appears to continue to run during the time that the virtual processor is ceded to the platform. The H_CEDE hcall() always returns to the next instruction, however, prior to executing the next instruction, any pending interrupt is taken. To simulate atomic testing for work, the H_CEDE call may be called with interrupts disabled, however, the H_CEDE call activates the virtual processor's MSR_{EE} bit to avoid going into a wait state with interrupts masked.

A multi-processor OS uses two methods to initiate work on one processor from another, in both cases the requesting processor places a unit of work structure on a queue, and then either signals the serving processor via an Inter-Processor interrupt to service the work queue, or waits until the serving processor polls the work queue. The former method translates directly to the SPLPAR environment, the second method may experience significant performance degradation if the serving processor has ceded. To provide a solution to this performance problem, the SPLPAR provides the H_PROD hcall(). The H_PROD hcall() takes as a parameter the virtual processor number of the serving processor. Waking a potentially ceded or ceding processor is subject to many race conditions. The semantic of the H_PROD hcall() attempts to minimize these race conditions. First the H_CEDE and H_PROD hcall()s serialize on each other per target virtual processor. Secondly by having the H_PROD firmware set a per virtual processor memory bit before attempting to determine if the target virtual processor is preempted. If the processor is not preempted the H_PROD hcall() immediately returns, else the processor is dispatched and the memory bit is reset. If the processor was dispatched, and subsequently the virtual processor does a H_CEDE operation, the H_CEDE hcall() checks the virtual processor's memory bit and if set, resets the bit and returns immediately (not ceding the physical processor to another

virtual processor). An OS might choose to always do an H_PROD after an enqueue to a polled queue or it might qualify making the H_PROD hcall() with a status bit set by the by the target processor when it decides to cede its virtual processor.

Locking in a SPLPAR environment presents a problem for multi-programming OSs, in that the virtual processor that is holding a lock may have been preempted. In that case, spinning, waiting for the lock, simply wastes time since the lock holder is in no position to release the lock -- it needs processor cycles and cannot get them for some period of time and the spinner is using up processor cycles waiting for the lock. The condition is known as a live lock, however, it eventually resolves itself. The SPLPAR optimization to alleviate this problem is to have the waiting virtual processor "confer" its processor cycles to the lock holder's virtual processor until the lock holder has had a chance to run another dispatch time slice.

As with the cede/prod pair of functions above, the confer function is subject to timing window races between the waiting process determining that the lock holder has been preempted and execution of the H_CONFER hcall() during which time the originally holding virtual processor may have been dispatched, released the lock and ceded the processor. To manage this situation, the H_CONFER takes two parameters, one that specifies the virtual processor(s) that are to receive the cycles and the second parameter (valid only when a single processor is specified) which represents the dispatch count of the holding virtual processor atomically captured when the waiting processor decided to confer its cycles to the waiting processor.

The semantic of H_CONFER checks the processor parameter for validity, then if it is the "all processors" code proceeds to the description below. If the processor parameter refers to a valid virtual processor owned by the calling virtual processor's partition, that is not dispatched, that has not conferred its cycles to all other processors, and who's current dispatch count matches that of the second parameter, the time remaining from the calling processors time slice is conferred to the specified virtual processor.

If the first parameter of H_CONFER specifies the "all processors" code, then it marks the calling virtual processor to confer all its cycles until all of the partition's virtual processors, that have not ceded or conferred their cycles, have had a chance to run a dispatch time slice. The "all processors" version may be viewed as having the hypervisor record the dispatch counts for all the other platform processors in the calling virtual processor's hypervisor owned "confer structure", then prior to any subsequent dispatch of the calling processor, if the confer structure is not clear, the hypervisor does the equivalent of removing one entry from the confer structure and calling H_CONFER for the specific virtual processor. If the specific virtual processor confer is rejected (because the virtual processor is running, ceded, conferred, or the dispatch count does not match) then the next entry is tried until the confer structure is clear before the originally calling virtual processor is re-dispatched.

Virtual processors may migrate among the physical processor pool from one dispatch cycle to the next. OF device tree properties that relate the characteristics of the specific physical processor such as location codes, and other vital product data cannot be consistent and are not reported in the nodes of type **`cpu`** if the partition is running in SPLPAR mode. Most processor characteristics properties such as time base increment rate, are consistent for all processors in the system physical and virtual so are still reported via their standard properties. Additionally nodes of type **`L2`** are not present in the tree since they are shared with other virtual processors making optimizations based upon their characteristics impossible. The Processor Identification Register (PIR) should not be accessed by the OS since from cycle to cycle the OS may get different readings, instead the virtual processor number (the number from the **`"ibm,ppc-interrupt-server#s"`** property, contained in the nodes of type **`cpu`**, associated with this virtual processor) is used as the processor number to be passed as parameters to RTAS and hcall() routines for managing interrupts etc.

Software Note: When the client program (OS) first gets control during the boot sequence, the virtual processor number of the single processor that is operational is identified by the **`/chosen`** node of the device tree. The **`cpu`** nodes list the other virtual processors that the first processor may start. These are started one at a time, giving the virtual processor number as an input parameter to the call. As each processor starts, it starts executing a program that picks up its virtual processor number from a memory structure initialized by the processor that called the start-cpu

function. The newly started processor then records the location of its per processor memory structure (where it saves its virtual processor number) in one of the SPRG registers.

14.11.1 Virtual Processor Areas

The per processor areas are registered with the H_REGISTER_VPA hcall() that takes three parameters. The first parameter is a flags field that specifies the specific sub function to be performed, the second is the virtual processor number of one of the processors owned by the calling virtual processor's partition for whom the area is being registered. The third parameter is the logical address, within the calling virtual processor's partition, of the contiguous logically addressed storage area to be registered. Registered areas are aligned on a cache line (l1) size boundary and may not span an LMB boundary and for the CMO option may not span an entitlement granule boundary. The length of the area is provided to the hcall() in starting in byte offset 4 of the area being registered. The H_REGISTER _VPA hcall() registers various types of areas, and after verifying the parameters, initializes the structure's variables.

Per Virtual Processor Area: This area contains shared processor operating parameters as defined in Table 184, "Per Virtual Processor Area," on page 449. A shared processor LPAR aware OS registers this area early in its initialization. The other types of virtual processor areas can only be registered after the Per Virtual Processor Area has been successfully registered. The minimum length of the Per Virtual Processor Area is 640 bytes and the structure may not span a 4096 byte boundary.

Dispatch Trace Log Buffer: This area is optionally registered by OS's that desire to collect performance measurement data relative to its shared processor dispatching. The minimum size of this area is 48 bytes while the maximum is 4 GB. See 14.11.1.2, "Dispatch Trace Log Buffer," on page 452 for more details

SLB Shadow Buffer: This area is optionally registered by OS's that support the SLB-shadow function set. The structure may not span a 4096 byte boundary. This function set allows the hypervisor to significantly reduce the overhead associated with virtual processor dispatch in a shared processor LPAR environment, and to provide enhanced recovery from SLB hardware errors. See 14.11.1.3, "SLB Shadow Buffer," on page 453 for more details.

Software Note: Registering, deregistering or changing the value of a variable in one of the Virtual Processor Areas for a different virtual processor (i.e. changing a value in the VPA of processor A from processor B) may be problematic. In no cases is partition integrity be compromised, but results may be imprecise if such a change is made during the virtual processor preempt/dispatch window. If the owning processor is started, registration or deregistration should only be done by the owning processor, if the processor is stopped, registration or deregistration can safely be done by other processors. Also, for example, changing the number of persistent SLB Shadow Buffer entries cause uncertainty in the number of currently valid SLB entries in that virtual processor. In some cases, such as turning on and off dispatch tracing, such uncertainty may be acceptable.

14.11.1.1 Per Virtual Processor Area

Table 184. Per Virtual Processor Area

Byte Offset	Length in Bytes	Variable Description
0x00	4	Descriptor: This field is supplied for OS identification use, it may be set to any value that may be useful (such as a pattern that may be identified in a dump) or it may be left uninitalized. Historic values include: 0xD397D781
0x04	2 (unsigned)	Size: The size of the registered structure (640)
0x6 - 0x17	18	Reserved for Firmware Use
0x18 - 0x1B	4	Physical Processor FRU ID

Table 184. Per Virtual Processor Area ***(Continued)***

Byte Offset	Length in Bytes	Variable Description
0x1C - 0x1F	4	Physical Processor on FRU ID
0x20 - 0x57	56	Reserved for Firmware Use
0x58 - 0x5F	8	Virtual processor home node associativity changes counters (changes in the 8 most important associativity levels)
0x60 - 0xAF	80	Reserved for Firmware Use
0xB0	1	Cede Latency Specifier
0xB1	7	Reserved For LoPAPR Expansion
0xB8	1	Dispatch Trace Log Enable Mask: (Note this entry is valid only if a Dispatch Trace Log Buffer has been registered). A Trace Log Entry is created when the virtual processor is dispatched following its preemption for an enabled cause. =0 no dispatch trace logging Bit 7 =1 Trace voluntary (OS initiated) virtual processor waits Bit 6 =1 Trace time slice preempts Bit 5 = 1 Trace virtual partition memory page faults. All other values are reserved
0xB9	1	Bits 0-6 Reserved Bit 7 = 0 -- Dedicated processor cycle donation disabled Bit 7 = 1 -- Dedicated processor cycle donation enabled.
0xBA	1	Maintain FPRs: =0 architected state of floating point registers may be discarded at any time, =1 architected state of floating point registers must be maintained, all other values are reserved Note: When set in conjunction with offset 0xFF the 128 bit VSX space is saved on processors supporting the VSX option (*Power ISA* [1] 2.06 and beyond).
0xBB	1	Maintain PMCs: =0 architected state of performance monitor counters may be discarded at any time, =1 architected state of performance monitor counters must be maintained, all other values are reserved
0xBC-0xD7	28	Reserved For Firmware Use
0xD8-0xDF	8	Any non-zero value is taken by the firmware to be the OS, estimate, in PURR units, of the cumulative number of cycles that it has consumed on this virtual processor, while idle, since it was initialized.
0xE0 - 0xFB	28	Reserved for Firmware Use
0xFC	2 (unsigned)	Maintain #SLBs: This number of Segment Lookaside Buffer Registers (up to the platform implementation maximum) are maintained, all others (up to the platform implementing maximum) may be discarded at any time. The value 0xFFFF maintains all SLBs
0xFE	1	Idle: =0 The OS is busy on this processor =1 The OS is idle on this processor All other values are reserved
0xFF	1	Maintain VMX state: =0 architected state of the processor's VMX facility, may be discarded at any time =1 architected state of the processor's VMX facility, must be maintained All other values are reserved Note: When set in conjunction with offset 0xBA the 128 bit VSX space is saved on processors supporting the VSX option (*Power ISA* [1] 2.06 and beyond).

Table 184. Per Virtual Processor Area ***(Continued)***

Byte Offset	Length in Bytes	Variable Description
0x100	4 (unsigned)	Virtual Processor Dispatch Counter: (Even when virtual processor is dispatched odd when it is preempted/ceded/conferred)
0x104	4 (unsigned)	Virtual Processor Dispatch Dispersion Accumulator: Incremented on each virtual processor dispatch if the physical processor differs from that of the last dispatch.
0x108	8 (unsigned)	Virtual Processor Virtual Partition Memory Fault Counter: Incremented on each Virtual Partition Memory page fault.
0x110	8 (unsigned)	Virtual Processor Virtual Partition Memory Fault Time Accumulator: Time, in Time Base units, that the virtual processor has been blocked waiting for the resolution of virtual Partition Memory page faults.
0x118 - 0x11F	8	Unsigned accumulation of PURR cycles expropriated by the hypervisor when VPA byte offset 0xFE = 1
0x120 - 0x127	8	Unsigned accumulation of SPURR cycles expropriated by the hypervisor when VPA byte offset 0xFE = 1
0x128 - 0x12F	8	Unsigned accumulation of PURR cycles expropriated by the hypervisor when VPA byte offset 0xFE = 0
0x130 - 0x137	8	Unsigned accumulation of SPURR cycles expropriated by the hypervisor when VPA byte offset 0xFE = 0
0x138 - 0x13F	8	Unsigned accumulation of PURR cycles donated to the processor pool when VPA byte offset 0xFE = 1
0x140 - 0x147	8	Unsigned accumulation of SPURR cycles donated to the processor pool when VPA byte offset 0xFE = 1
0x148 - 0x14F	8	Unsigned accumulation of PURR cycles donated to the processor pool when VPA byte offset 0xFE = 0
0x150 - 0x157	8	Unsigned accumulation of SPURR cycles donated to the processor pool when VPA byte offset 0xFE = 0
0x158-0x15F	8	Accumulated virtual processor wait interval 3 timebase cycles. (waiting for physical processor availability)
0x160 – 0x167	8	Accumulated virtual processor wait interval 2 timebase cycles. (waiting for virtual processor capacity)
0x168 – 0x16F	8	Accumulated virtual processor wait interval 1 timebase cycles. (waiting for virtual processor ready to run)
0x170 - 0x177	8	Reserved for Firmware Use
0x178 - 0x17F	8	Reserved for Firmware Use
0x180 – 0x183	4	For the CMO option: The OS may report in this field as a hint to the hypervisor the accumulated number, since the virtual processor was started, of ‘page in’ operations initiated for pages that were previously swapped out.”
0x184 – 0x187	4	Reserved for Firmware Use
0x188 – 0x18F	8	Reserved for Firmware Use
0x190 – 0x197	8	.Reserved for Firmware Use
0x198 – 0x217	128	Reserved for Firmware Use
0x218 - 0x21F	8	Dispatch Trace Log buffer index counter.
0x220 - 0x27F	96	Reserved for Firmware Use

R1–14.11.1.1–1. For the SPLPAR option: If the OS registers a Per Virtual Processor Area, it must correspond to the format specified in Table 184, “Per Virtual Processor Area,” on page 449.

14.11.1.2 Dispatch Trace Log Buffer

The optional virtual processor dispatch trace log buffer is a circularly managed buffer containing individual 48 byte entries, with the first entry starting at byte offset 0. Therefore, the 4 byte registration size field is overwritten by the first Trace Log Buffer entry. (Note the hypervisor rounds down the dispatch trace log buffer length to a multiple of 48 bytes and wraps when reaching that boundary.) A vpa location Table 184, “Per Virtual Processor Area,” on page 449 contains the index counter that the hypervisor increments each time that it makes a dispatch trace log entry such that it always indicates the next entry to be filled. The low order bits (modulo the buffer length divided by 48) of the counter provide the index of the next entry to be filled, therefore, the buffer wraps each (buffer length divided by 48 entries), while the high order counter bits indicate how many buffer wraps have occurred. Prior to enabling dispatch trace logging, the OS should initialize the vpa index counter to the value of 0. The format of dispatch trace log buffer entries is given in Table 185, “Dispatch Trace Log Buffer Entry,” on page 452.

The architectural intent is that OS trace tools keep a shadow index counter into the log buffer of the next entry to be filled by the hypervisor. Prior to making an entry of their own, such tools compare their index counters with that of the hypervisor from the vpa, if they are equal, no preempts/dispatches have occurred since the last OS trace hook. If the two index counters are not equal, then the OS trace tool processes the intermediate time stamps into the OS’s trace log, updating its dispatch trace log buffer index until all have been processed, then the new trace entry is added to the OS’s trace log. Note, because of races, the processor may be preempted just prior to the OS trace tool adding the new trace log entry, to handle this case, the OS trace tool can examine the dispatch trace log buffer index immediately after the adding of the new trace log entry and if needed adjust its own trace log. In the extremely unlikely event that the two counters are off by trace buffer length divided by forty eight or more counts, the OS trace tool can detect that a dispatch trace log buffer overflow has occurred, and trace data has been lost.

Table 185. Dispatch Trace Log Buffer Entry

Byte Offset	Length in Bytes	Variable Description
0x0	1	Reason Code for the virtual processor dispatch: 0: The virtual processor was dispatched at the external interrupt vector location to handle an IOA interrupt, Virtual interrupt, or interprocessor interrupt. 1: The virtual processor was dispatched to handle firmware internal events. 2: The virtual processor was dispatched at the next sequential instruction due to an H_PROD call by another partition processor. 3: The virtual processor was dispatched at the DECR interrupt vector due to a decrementer interrupt. 4: The processor was dispatched at location specified in load module (boot) or at the system reset interrupt vector. (virtual yellow button). 5: The virtual processor was dispatched to handle firmware internal events 6: The virtual processor was dispatched at the next sequential instruction to use cycles conferred from another partition processor 7: The virtual processor was dispatched at the next sequential instruction for its entitled time slice. 8: The virtual processor was dispatched at the faulting instruction following a virtual partition memory page fault.
0x1	1	Reason Code for virtual processor preemption: 0: Not used (for compatibility with earlier versions of the facility) 1: Firmware internal event 2: Virtual processor called H_CEDE 3: Virtual processor called H_CONFER 4: Virtual processor reached the end of its timeslice (HDEC) 5: Partition Migration/Hibernation page fault 6: Virtual memory page fault
0x2 - 0x3	2	Processor index of the physical processor actualizing the thread on this dispatch.
0x4 - 0x7	4	Time Base Delta between enqueued to dispatcher and actual dispatch on a physical processor
0x8 - 0xB	4	Time Base Delta between ready to run and enqueue to dispatcher

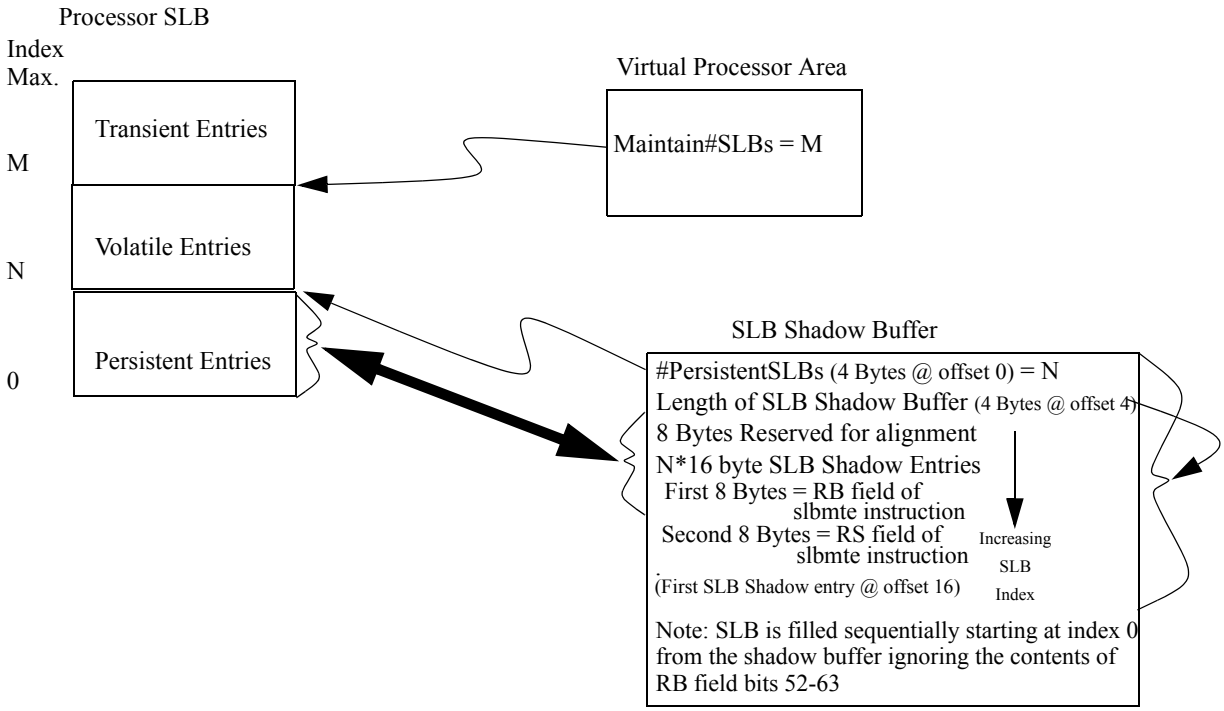
Table 185. Dispatch Trace Log Buffer Entry

Byte Offset	Length in Bytes	Variable Description
0xC - 0xF	4	Time Base Delta between waiting and ready to run (preempt/fault resolution time)
0x10 - 0x17	8	Time Base Value at the time of dispatch/wait
0x18 - 0x1F	8	For virtual processor preemption reason codes 5 & 6: Logical real address of faulting page; else reserved.
0x20 - 0x27	8	SRR0: At the time of preempt/wait
0x28 - 0x2F	8	SRR1: At the time of preempt/wait

R1–14.11.1.2–1. For the SPLPAR option: If the OS registers a Dispatch Trace Log Buffer, it must correspond to the format specified in Table 185, “Dispatch Trace Log Buffer Entry,” on page 452.

14.11.1.3 SLB Shadow Buffer

On platforms supporting the SLB-Buffer function set, the OS may optionally register an SLB shadow buffer area. When the OS takes this option, it allows the hypervisor to optimize the saving of SLB entries, thus reducing overhead and providing more processor capacity for the OS, and also allows the platform to recover from certain SLB hardware faults. When the OS registers an SLB shadow buffer for its virtual processor, the processor’s SLB is architecturally divided into three categories relative to their durability as depicted in Figure , “,” on page 453.



OS may dynamically change M and N (for (N+1)*16 <= Length of SLB Shadow Buffer)

Figure 13. Processor SLB relationship to the OS registered VPA and SLB Shadow Buffer

Each category of SLB entries consists of 0-n contiguous SLBs.

Persistent Entries: The first N (starting at SLB index 0, N specified by the numeric content of the first 4 bytes of the registered SLB Shadow Buffer) SLBs are maintained persistent across all virtual processor dispatches unless an unrecovered SLB error is noted. OS maintains a shadow copy of those SLB entries in the registered SLB shadow buffer. The OS sizes its SLB Shadow buffer for the largest number of persistent entries it can ever maintain. If the OS registers an SLB Shadow buffer, the hypervisor does not save the contents of the Persistent entries on virtual processor preempt, cede, or confer. The OS should minimally record as persistent the entries it needs to handle its SLB fault interrupts to fill in required Volatile (and potentially) Transient entries.

Volatile Entries: The next M-N SLBs (beginning at the next higher SLB index after the last Persistent entry up through the entry specified by the "maintain#SLBs" parameter of the VPA) may disappear. The OS needs to be prepared to recover these entries via SLB fault interrupts. For performance optimization, the hypervisor normally maintains the state of these entries across H_DECR interrupts and most hcalls(), they may be lost on H_CEDE calls.

Transient Entries: The platform makes no attempt to maintain the state of these entries and they may be lost at any time.

The OS may dynamically change the number of Persistent entries by atomically changing the value of the 4 byte parameter at SLB Shadow Buffer offset 0.

The hypervisor does not explicitly check the value of this parameter, however, the hypervisor limits the number of SLBs that it attempts to load from the shadow buffer to the lesser of the maximum number of SLB entries implemented by the platform, or the maximum number of entries containable in the SLB Shadow buffer length when it was registered.

R1–14.11.1.3–1. For the SPLPAR option: If the OS registers an SLB Shadow Buffer, it must correspond to the format specified in Figure 13, "Processor SLB relationship to the OS registered VPA and SLB Shadow Buffer," on page 453.

14.11.2 Shared Processor LPAR OF Extensions

14.11.2.1 Shared Processor LPAR Function Sets in "ibm,hypertas-functions"

1. hcall-splpar
2. hcall-pic
3. SLB-Buffer

14.11.2.2 Device Tree Variances

If an SPLPAR implementation does not maintain a fixed relationship between the virtual processor that it reports to the OS image in the OF device tree properties and the physical processor that it uses to actualize the virtual processor, then OF entities that imply a fixed physical relationship are not reported. These may include those listed in Table 186, "OF Variances due to SPLPAR," on page 455.

Table 186. OF Variances due to SPLPAR

Entity	Variance to standard definition
"ibm,loc-code" property	If the physical relationship between virtual processors and physical processors is not constant this property is omitted from the virtual processor's node. If missing, the OS should not run diagnostics on the virtual processor
"l2-cache" property	If the physical relationship between virtual processors and physical processors is not constant the secondary cache characteristics are not relevant and this property is omitted from the virtual processor's node.
Nodes named **l2-cache**	If the physical relationship between virtual processors and physical processors is not constant the secondary cache characteristics are not relevant and this node is omitted from the partition's device tree.
"ibm,associativity" property	If the physical relationship between virtual processors and physical processors is not constant the "**ibm,associativity"** property reflects the same domain for all virtual processors actualized by a given physical processor pool. Note, even though the associativity of virtual processors may be indistinguishable, the associativity among other platform resources may be relevant.

R1–14.11.2.2–1. For the SPLPAR option: If the platform does not maintain a fixed relationship between its virtual processors and the physical processors that actualize them, then the platform must vary the device tree elements as outlined in Table 186, "OF Variances due to SPLPAR," on page 455.

14.11.3 Shared Processor LPAR Hypervisor Extensions

14.11.3.1 Virtual Processor Preempt/Dispatch

A new virtual processor is dispatched on a physical processor when one of the following conditions happens:

- The physical processor is idle and a virtual processor was made ready to run (interrupt or prod)
- The old virtual processor exhausted its time slice (HDECR interrupt).
- The old virtual processor ceded/conferred its cycles.

When one of the above conditions occurs, the hypervisor, by default, records all the virtual processor architected state including the Time Base and Decrementer values and sets the hypervisor timer services to wake the virtual processor per the setting of the decrementer. The virtual processor's Processor Utilization Register value for this dispatch is computed. The VPA's dispatch count is incremented (such that the result is odd). Then the hypervisor selects a new virtual processor to dispatch on the physical processor using an implementation dependent algorithm having the following characteristics given in priority order:

1. The virtual processor is "ready to run" (has not ceded/conferred its cycles or exhausted its time slice).
2. Ready to run virtual processors are dispatched prior to waiting in excess of their maximum specified latency.
3. Of the non-latency critical virtual processors ready to run, select the virtual processor that is most likely to have its working set in the physical processor's cache or for other reasons runs most efficiently on the physical processor.

If no virtual processor is "ready to run" at this time, start accumulating the Pool Idle Count (PIC) of the total number of idle processor cycles in the physical processor pool.

Optionally, flags in the VPA may be set by the OS to indicate to the hypervisor that selected architected state of the virtual processor need not be maintained (that is, the contents of these architected facilities may be lost at any time without notice). The hypervisor may then optimize its preempt/dispatch routines accordingly. Refer to Table 184, "Per Virtual Processor Area," on page 449 and SLB Shadow Buffer description for the definition of these flags and values.

The hypervisor modifies any such OS setable and readable processor state that is not explicitly saved and restored on a virtual processor dispatch so as to prevent a covert channel between partitions.

When the virtual processor is dispatched, the virtual processor's "prod" bit is reset, the saved architected state of the virtual processor is restored from that saved when the virtual processor was preempted, ceded, or conferred, except for the time base which retains the current value of the physical processor and the decrementer which is reduced from the state saved value per current Time Base value minus saved Time Base value. The hypervisor sets up for computing the PUR value increment for the dispatch.

At this time, the hypervisor increments the virtual processor's VPA dispatch count (such that the value is even). The hypervisor checks the VPA's dispatch log flag, if set, the hypervisor creates a pair of log entries in the dispatch log and stores the circular buffer index in the first buffer entry.

If the virtual processor was signaled with an interrupt condition and the physical interrupt has been reset, then the hypervisor adjusts the virtual processor architected state to reflects that of a physical processor taking the same interrupt prior to executing the next sequential instruction and execution starts with the first instruction in the appropriate interrupt vector. If no interrupt has been signaled to the virtual processor or the physical interrupt is still active, then execution starts at the next sequential instruction following the instruction as noted by the hypervisor when the virtual processor ceded, conferred, or was preempted.

The Platform allocates processor capacity to a partition's virtual processors using the architectural metaphor of a "dispatch wheel" with a fixed implementation dependent rotation period. Each virtual processor receives a time slice each rotation of the dispatch wheel. The length of the time slice is determined by a number of parameters, the OS image has direct control, within constraints, over three of these parameters (number of virtual processors, Entitled Processor Capacity Percentage, Variable Processor Capacity Weight). The constraints are determined by partition and partition aggregate configurations that are outside the scope of this architecture. For reference, partition definitions provide the initial settings of these parameters while the aggregation configurations provide the constraints (including the degenerate case where an aggregation encapsulates only a single member LPAR).

Entitled Processor Capacity Percentage:The percentage of a physical processor that the hypervisor guarantees to be available to the partition's virtual processors (distributed in a uniform manner among the partition's virtual processors -- thus the number of virtual processors affects the time slice size) each dispatch cycle. Capacity ceded or conferred from one partition virtual processor extends the time slices offered to other partition processors. Capacity ceded or conferred after all of the partition's virtual processors have been dispatch is added to the variable capacity kitty. The initial, minimum and maximum constraint values of this parameter are determined by the partition configuration definition. The H_SET_PPP hcall() allows the OS image to set this parameter within the constraints imposed by the partition configuration definition minimum and maximums plus constraints imposed by partition aggregation.

Variable Processor Capacity Weight:The unitless factor that the hypervisor uses to assign processor capacity in addition to the Entitled Processor Capacity Percentage. This factor may take the values 0 to 255. A virtual processor's time slice may be extended to allow it to use capacity unused by other partitions, or not needed to meet the Entitled Processor Capacity Percentage of the active partitions. A partition is offered a portion of this variable capacity kitty equal to: (Variable Processor Capacity Weight for the partition) / (summation of Variable Processor Capacity Weights for all competing partitions). The initial value of this parameter is determined by the partition configuration definition. The H_SET_PPP hcall() allows the OS image to set this parameter within the constraints imposed by the partition configuration definition maximum. Certain partition definitions may not allow any variable processor capacity allocation.

Unallocated Processor Capacity Percentage: The amount of processor capacity that is currently available within the constraints of the LPAR's current environment for allocation to Entitled Processor Capacity Percentage. Race conditions may change the current environment before a request for this capacity can be performed, resulting in a constrained return from such a request.

Unallocated Variable Processor Capacity Weight: The amount of variable processor capacity weight that is currently available within the constraints of the LPAR's current environment for allocation to the partition's variable processor capacity weight. Race conditions may change the current environment before a request for this capacity can be performed, resulting in a constrained return from such a request.

System Parameters readable via the *ibm,get-system-parameter* RTAS call (see Section 7.3.16.1, “ibm,get-system-parameter,” on page 211) communicate a variety of configuration and constraint parameters among which are determined by the partition definition.

By means that are beyond the scope of this architecture, various partitions may be organized into aggregations, for example “LPAR groups”, for the purposes of load balancing. These aggregations may impose constraints such as: “The summation of the minimum available capacity for all virtual processors supported by the LPAR group cannot exceed 100% of the group’s configured capacity”.

R1–14.11.3.1–1. For the SPLPAR option: The platform must dispatch each partition virtual processors each dispatch cycle unless prevented by the semantics of the H_CONFER hcall().

R1–14.11.3.1–2. For the SPLPAR option: The summation of the processing capacity that the platform dispatches to the virtual processors of each partition must be at least equal to that partition's Entitled Processor Capacity Percentage unless prevented by the semantics of the H_CONFER and H_CEDE hcall()s.

R1–14.11.3.1–3. For the SPLPAR option: The processing capacity that the platform dispatches to each of the partition's virtual processors must be substantially equal unless prevented by the semantics of the H_CONFER and H_CEDE hcall()s.

R1–14.11.3.1–4. For the SPLPAR option: The platform must distribute processor capacity allocated to SPLPAR virtual processor actualization not consumed due to Requirements R1–14.11.3.1–1, R1–14.11.3.1–2, and R1–14.11.3.1–3 to partitions in strict accordance with the definition of Variable Processor Capacity Weight unless prevented by the LPAR's definition (capped) or the semantics of the H_CONFER and H_CEDE hcall()s.

NOTE: A value of 0 for a Variable Processor Capacity Weight effectively caps the partition at its Entitled Processor Capacity Percentage value.

R1–14.11.3.1–5. For the SPLPAR option on platforms: The platform must increment the counters in VPA offsets 0x158-0x16F per their definitions in Section 184, “Per Virtual Processor Area,” on page 449.

R1–14.11.3.1–6. For the SPLPAR option on platforms : To maintain compatibility across partition migration and firmware version levels the OS must be prepared for platform implementations that do not increment VPA offsets 0x158 – 0x16F.

14.11.3.2 H_REGISTER_VPA

Register Virtual Processor Areas (these include the parameter area known as the VPA, the Dispatch Trace Log Buffer, and if the SLB-Buffer function set is supported, the SLB Shadow Buffer). Note if the caller makes multiple registration requests for a given per virtual processor area for a given virtual processor, the last registration wins, and if the same memory area is registered for multiple processors, the area contents are unpredictable, however, LPAR isolation is not compromised.

The syntax of the H_REGISTER_VPA hcall() is given below.

```
int64                   /* H_Success,
                        /* H_Parameter,
                        /* H_RESCINDED: A specified parameter refers to a rescinded shared logical resource/
                        /* H_RESOURCE -- a required resource was not available */
                                        /* (probable cause is registering a trace buffer without a VPA */
```

```
                                        /* or deregistering a VPA with a registered trace buffer) */
                    /* H_Hardware -- a hardware event prevented operation */
                    /* H_MLENGTH_PARM: For the CMO option, the requested area to be registered crossed a */
                                        /* memory entitlement granule boundary */
hcall (const unit64 H_REGISTER_VPA,     /* Register the specified per virtual Processor Area */
        uint64      flags               /* The sub functions for this hcall() are encoded in bits 16-18 */
                                        /* 000 = Reserved */
                                        /* 001 = Register Virtual Processor Area */
                                        /* 010 = Register Dispatch Trace Log Buffer*/
                                        /* 011 = Register SLB Shadow Buffer (if SLB-Buffer function */
                                        /*       set is supported) */
                                        /* 100 = Reserved */
                                        /* 101 = Deregister Virtual Processor Area */
                                        /* 110 = Deregister Dispatch Trace Log Buffer */
                                        /* 111 = Reserved */
                                        /* 111 = Deregister SLB Shadow Buffer (if SLB-Buffer /*
                                        /*       function set is supported) */
        uint64      proc-no,            /* Virtual Processor Number */
        uint64      vpa);               /* Logical Address of the VPA being registered */
```

Semantics:

- Verify that the flags parameter is a supported value else return H_Parameter. (That the subfunction field (Bits 16-23) is one of the values supported by this call. Optionally that all other bits are zero. Callers should not set any bits other than those specifically defined, however, implementations are not required to check the value of bits outside of the subfunction field.)
- Verify that the proc-no parameter references a virtual processor owned by the calling virtual processor's partition else return H_Parameter
- If the sub function is a register, verify that the addr parameter is an L1 cache line aligned logical address within the memory owned by the calling virtual processor's partition else return H_Parameter.
  - If the Shared Logical Resource option is implemented and the addr parameter represents a shared logical resource location that has been rescinded by the owner, return H_RESCINDED.
- Case on subfunction in flags parameter:
- Register VPA:
  - Verify that the size field (2 bytes) at offset 0x4 is at least 640 bytes else return H_Parameter.
  - Verify that the entire structure (per the size field and vpa) does not span a 4096 byte boundary else return H_Parameter.
  - Record the specified processor's vpa logical address for access by other SPLPAR hypervisor functions.
  - Initialize the contents of the area per Section 14.11.1, "Virtual Processor Areas," on page 449.
  - Return H_Succes
- Register Dispatch Trace Log Buffer:
  - Verify that the size field (4 bytes) at offset 0x4 is at least 48 bytes else return H_Parameter.
  - For the CMO option, verify that the entire structure (per the size field and vpa parameter) does not span a memory entitlement granule boundary else return H_MLENGTH_PARM.

- Verify that a VPA has been registered for the specified virtual processor else return H_RESOURCE.
- Initialize the specified processor's preempt/dispatch trace log buffer pointers and index.
- Return H_Success.

- Register SLB Shadow Buffer (if SLB-Buffer function set is supported):
  - Verify that the size field (4 bytes) at offset 0x4 is at least 8 bytes and that the entire structure (per the size and vpa parameters) does not span a 4096 byte boundary else return H_Parameter.
  - Verify that a VPA has been registered for the specified virtual processor else return H_RESOURCE.
  - Initialize the specified processor's SLB Shadow buffer pointers and set the maximum persistent SLB restore index to the lesser of the maximum number of processor SLBs or the maximum number of entries in the registered SLB Shadow buffer.
  - Return H_Success.
- Deregister VPA:
  - Verify that a Dispatch Trace Log buffer is not registered for the specified processor else return H_RESOURCE.
  - Verify that an SLB Shadow buffer is not registered for the specified processor else return H_RESOURCE.
  - Clear any partition memory pointer to the specified processor's VPA (note no check is made that a valid VPA registration exists).
  - Return H_Success.
- Deregister Dispatch Trace Log Buffer:
  - Clear any partition memory and/ or hypervisor pointer to the specified processor's Dispatch Trace Buffer (note no check is made that a valid Dispatch Trace Buffer registration exists).
  - Return H_Success.
- Deregister SLB Shadow Buffer (if SLB-Buffer function set is supported):
  - Clear any hypervisor pointer(s) to the specified processor's SLB Shadow buffer (note no check is made that a valid SLB Shadow buffer registration exists).
  - Zero the hypervisor's maximum persistent SLB restore index for the specified processor.
  - Return H_Success.
- Else Return H_Function.

R1–14.11.3.2–1. For the SPLPAR option: The platform must implement the H_REGISTER_PVA hcall() following the syntax and semantics of Section 14.11.3.2, "H_REGISTER_VPA," on page 457.

R1–14.11.3.2–2. For the SLPAR plus SLB Shadow Buffer options: The platform must register, and deregister the optional SLB Shadow buffer per the syntax and semantics of Section 14.11.3.2, "H_REGISTER_VPA," on page 457.

R1–14.11.3.2–3. For the SLPAR plus SLB Shadow Buffer options: The platform must make persistent the SLB entries recorded by the OS within the SLB Shadow buffer as described in 14.11.1.3, "SLB Shadow Buffer," on page 453.

14.11.3.3 H_CEDE

The architectural intent of this hcall() is to have the virtual processor, which has no useful work to do, enter a wait state ceding its processor capacity to other virtual processors until some useful work appears, signaled either through an interrupt or a prod hcall(). To help the caller reduce race conditions, this call may be made with interrupts disabled but the semantics of the hcall() enable the virtual processor's interrupts so that it may always receive wake up interrupt signals. As a hint to the hypervisor, the cede latency specifier Table 184, "Per Virtual Processor Area," on page 449 indicates how long the OS can tolerate the latency to an H_PROD hcall() or interrupt, this may affect how the hypervisor chooses to use or even power down the actualizing physical processor in the mean time.

Software Note: The floating point registers may not be preserved by this call if the "Maintain FPRs" field of the VPA =0, see Table 184, "Per Virtual Processor Area," on page 449

Syntax:

```
int64          /* H_Success:     Expected return code upon return when more work is potentially available */
               /* H_Hardware:    The hcall() experienced a hardware fault potentially preventing the function */
    hcall (const uint64 H_CEDE);          /*Cede the calling virtual processor's cycles to the platform */
```

Semantics:

- Enable the virtual processor's MSR_{EE} bit as if it was on at the time of the call.
- Serialize for the virtual processor's control structure with H_PROD.
- If the virtual processor's "prod" bit is set, then:
  - Reset the virtual processor's "prod" bit.
  - Release the virtual processor's control structure.
  - Return H_Success.
- Record all the virtual processor architected state including the Time Base and Decrementer values.
- Set hypervisor timer services to wake the virtual processor per the setting of the decrementer.
- Mark the virtual processor as non-dispatchable until the processor is the target of an interrupt (system reset, external including decrementer or IPI) or PROD.
- Cede the time remaining in the virtual processor's time slice preferentially to the virtual processor's partition.
- Release the virtual processor's control structure.
- Dispatch some other virtual processor
- Return H_Success.

R1–14.11.3.3–1. For the SPLPAR option: The platform must implement the H_CEDE hcall() following the syntax and semantics of Section 14.11.3.3, "H_CEDE," on page 460.

14.11.3.4 H_CONFER

The architectural intent of this hcall() is to confer the callers processor capacity to the holder of a lock or the initiator of an event that the caller is waiting upon. If the caller knows the identity of the lock holder then the holder's virtual processor number is supplied as a parameter, if the caller does not know the identity of the lock holder then the "all processors" value of the proc parameter is specified. If the caller is conferring to the initiator of an event the proc parameter value of the calling processor. This call may be made with interrupts enabled or disabled. This call provides a reduced "kill set" of volatile registers, GPRs r0 and r4-r13 are preserved.

Software Note: The floating point registers may not be preserved by this call if the "Maintain FPRs" field of the VPA =0, see Table 184, "Per Virtual Processor Area," on page 449

Syntax:

```
int64          /* H_Success:     Expected return code upon return when more work is potentially available */
               /* H_Parameter:   The specified processor is not owned by the partition */
               /* H_Hardware:    The hcall() experienced a hardware fault potentially preventing the function */
    hcall (const uint32 H_CONFER,     /*Confer the calling virtual processor's cycles to the specified processor*/
               int32    proc,         /*Target Processor number -- minus 1 is all partition processors */
               uint32   dispatch);    /* The dispatch number (ignored if proc=caller) */
```

Semantics:

- Validate the proc number else return H_Parameter. Valid Values:
  -1 (all partition processors)
  0 through N one of the processor numbers of the calling processor's partition
  The calling processor's number forces a confer until the calling processor is PRODed
- If the proc number is for a single processor and the single processor is not the calling processor, then
  - If the dispatch parameter is not equal to the specified processor's hypervisor copy of the dispatch number or the hypervisor copy of the dispatch number is even, then return H_Success.
  - If the target processor has conferred its cycles to all others, then return H_Success.

  Firmware Implementation Note: If one were to confer to a processor that had conferred to all, then a dead lock could occur, however, there are valid cases with nested locks were this could happen, therefore, the hypervisor call silently ignores the confer.
- Record all the virtual processor architected state including the Time Base and Decrementer values.
- If the MSR_{EE} bit is on, set hypervisor timer services to wake the virtual processor per the setting of the decrementer.
- Mark the virtual processor as non-dispatchable until one of the following:
  - System reset interrupt.
  - The MSR_{EE} bit is on and the virtual processor is the target of an external interrupt (including decrementer or IPI).
  - The virtual processor is the target of a PROD operation.
  - The specified target processor (or all partition processors if the proc parameter value is a minus 1) have had the opportunity of a dispatch cycle.
- Confer the time remaining in the virtual processor's time slice to the virtual processor's partition.
- Dispatch the/a partition target virtual processor.
- Return H_Success.

R1–14.11.3.4–1. For the SPLPAR option: The platform must implement the H_CONFER hcall() following the syntax and semantics of Section 14.11.3.4, "H_CONFER," on page 460.

R1–14.11.3.4–2. For the SPLPAR option: The platform must implement the H_CONFER hcall() such that the only GPR that is modified by the call is r3.

14.11.3.5 H_PROD

Awakens the specific processor. This call provides a reduced “kill set” of volatile registers, GPRs r0 and r4-r13 are preserved.

Syntax:

```
int64          /* H_Success:     Expected return code upon return when more work is potentially available */
               /* H_Parameter:   The specified processor is not owned by the partition */
               /* H_Hardware:    The hcall() experienced a hardware fault potentially preventing the function */
    hcall (const uint64 H_PROD,        /*Mark the target processor runable */
               int64    proc);         /*Target Processor number */
```

Semantics:

- Verify that the target virtual processor specified by the proc parameter is owned by the calling virtual processor’s partition.
- Serialize for the Target Virtual Processor’s control structure with H_CEDE.
- Set “prod” bit in the target virtual processor’s control structure.
- If the target virtual processor is not ready to run, mark the target virtual processor ready to run.
- Release the target virtual processor’s control structure.
- Return H_Success.

R1–14.11.3.5–1. For the SPLPAR option: The platform must implement the H_PROD hcall() following the syntax and semantics of Section 14.11.3.5, “H_PROD,” on page 462.

R1–14.11.3.5–2. For the SPLPAR option: The platform must implement the H_PROD hcall() such that the only GPR that is modified by the call is r3.

14.11.3.6 H_GET_PPP

This hcall() returns the partition’s performance parameters. The parameters are packed into registers:

- Register R4 contains the Entitled Processor Capacity Percentage for the partition. In the case of a dedicated processor partition this value is 100* the number of processors owned by the partition.
- Register R5 contains the Unallocated Processor Capacity Percentage for the calling partition’s aggregation.
- Register R6 contains the aggregation numbers of up to 4 levels of aggregations that the partition may be a member.
  - Bytes 0-1: Reserved for future aggregation definition, and set to zero -- in the future this field may be given meaning.
  - Bytes 2-3: Reserved for future aggregation definition, and set to zero -- in the future this field may be given meaning.
  - Bytes 4-5: 16 bit binary representation of the “Group Number”.
  - Bytes 6-7: 16 bit binary representation of the “Pool Number”. In the case of a dedicated processor partition the “Pool Number” is not applicable which is represented by the code 0xFFFF.
- Register R7 contains the platform resource capacities:

- Bytes 0 Reserved for future platform resource capacity definition, set to zero -- in the future this field may be given meaning.
- Byte 1 is a bit field representing the capping mode of the partition's virtual processor(s):
  - Bits 0-6 are reserved, and set to zero -- in the future these bits may be given meaning as new capping modes are defined
  - Bit 7 -- The partition's virtual processor(s) are capped at their Entitled Processor Capacity Percentage. In the case of dedicated processors this bit is set.
- Byte 2: Variable Processor Capacity Weight. In the case of a dedicated processor partition this value is 0x00.
- Byte 3: Unallocated Variable Processor Capacity Weight for the calling partition's aggregation.
- Bytes 4-5 16 bit unsigned binary representation of the number of processors active in the caller's Processor Pool. In the case of a dedicated processor partition this value is 0x00.
- Bytes 6-7 16 bit binary representation of the number of processors active on the platform.

- When the value of the **"ibm,partition-performance-parameters-level"** (see Table B.6.2.1, "Root Node Properties," on page 673) is >=1 then register R8 contains the processor virtualization resource allocations. In the case of a dedicated processor partition R8 contains 0:
  - Bytes 0-1: 16 bit unsigned binary representation of the number of physical platform processors allocated to processor virtualization.
  - Bytes 2-4: 24 bit unsigned binary representation of the maximum processor capacity percentage that is available to the partition's pool.
  - Bytes 5-7: 24 bit unsigned binary representation of the entitled processor capacity percentage available to the partition's pool.

Syntax:

```
int64          /* H_Success:     Expected return code */
               /* H_Hardware:    The hcall() experienced a hardware fault potentially preventing the function */
  hcall (const uint64 H_GET_PPP);     /*Returns in R4 - R7 the Platform Performance Parameters. */
```

Semantics:

- Place the partition's performance parameters for the calling virtual processor's partition into the respective registers:
  - R4: The calling partition's Entitled Processor Capacity Percentage
  - R5: The calling partition's aggregation's Unallocated Processor Capacity Percentage.
  - R6: The aggregation numbers
  - R7: The platform resource capacities
  - R8: When "ibm,partition-performance-parameters-level" is >= 1 in the device tree, R8 is loaded with the processor virtualization resource allocations
  - Return H_Success.

R1–14.11.3.6–1. For the SPLPAR option: The platform must implement the H_GET_PPP hcall() following the syntax and semantics of Section 14.11.3.6, "H_GET_PPP," on page 462.

14.11.3.7 H_SET_PPP

This hcall() allows the partition to modify its entitled processor capacity percentage and variable processor capacity weight within limits. If one or both request parameters exceed the constraints of the calling LPAR's environment, the hypervisor limits the set value to the constrained value and returns H_Constrained. The H_GET_PPP call may be used to determine the actual current operational values. By the hypervisor constraining the actual values, the calling partition does not need special authority to make the H_SET_PPP hcall().

See Section 14.11.3.1, "Virtual Processor Preempt/Dispatch," on page 455 for definitions of these values.

Syntax:

```
int64          /* H_Success:      Expected return code */
               /* H_Parameter:    One or both the input parameters were invalid*/
               /* H_Constrained: One or both of the input parameters exceeded the partition's constraints*/
               /* H_Hardware:    The hcall() experienced a hardware fault potentially preventing the function */
    hcall (const uint64 H_SET_PPP,     /* Modifies the specified partition's performance parameters*/
               uint64   entitled,      /* Entitled Processor Capacity Percentage*/
               uint8    variable);     /* Variable Processor Capacity Weight*/
```

Semantics:

- Verify that the variable processor capacity weight is between 0 and 255 else return H_Parameter.
- Verify that the capacities specified is within the constraints of the partition:
  - If yes, atomically set the partition's entitled and variable capacity per the request and return H_Success.
  - If not set the partition's entitled and variable capacity as constrained by the partition's configuration and return H_Constrained.

Firmware Implementation Note: If the dispatch algorithm requires that the summation of variable capacities be updated, it is atomically updated with the set of the partition's weight.

R1–14.11.3.7–1. For the SPLPAR option: The platform must implement and make available to selected partitions, the H_SET_PPP hcall() following the syntax and semantics of Section 14.11.3.7, "H_SET_PPP," on page 464.

14.11.3.8 H_PURR

The Processor Utilization of Resources Register (PURR) is compatibly read through the H_PURR hcall(). In those implementations running on processors that do not implement the register in hardware, firmware simulates the function. On platforms that present the property **"ibm,rks-hcalls"** with bit 2 set (see Appendix B.6.3.1, "RTAS Node Properties," on page 690), this call provides a reduced "kill set" of volatile registers, GPRs r0 and r5-r13 are preserved.

Syntax:

```
uint64         /* H_Success:      Expected return code */
               /* H_Hardware:    The hcall() experienced a hardware fault potentially preventing the function */
    hcall (const uint64 H_PURR);       /*Returns in R4 the value of the Processor Utilization Register */
```

Semantics:

- If the platform presents the **"ibm,rks-hcall"** property with bit 2 set; then honor a kill set of volatile registers r3 & r4.
- Compute the PURR value for the calling virtual processor up to the current point in time and place in R4

- Return H_Success.

R1–14.11.3.8–1. For the SPLPAR option: The platform must implement the H_PURR hcall() following the syntax and semantics of Section 14.11.3.9, “H_POLL_PENDING,” on page 465.

14.11.3.9 H_POLL_PENDING

Certain implementations of the hypervisor steal processor cycles to perform administrative functions in the background. The purpose of the H_POLL_PENDING hcall() is to provide a OS, running atop such an implementation, with a hint of pending work so that it may more intelligently manage use of platform resources. The use of this call by an OS is totally optional since such an implementation also uses hardware mechanisms to ensure that the required cycles can be transparently stolen. It is assumed that the caller of H_POLL_PENDING is idle, if all threads of the processor are idle (as indicated by the idle flag at byte offset 0xFE of Table 184, “Per Virtual Processor Area,” on page 449), the hypervisor may choose to perform a background administrative task. The hypervisor returns H_PENDING if there is pending administrative work, at the time of the call, that it could dispatch to the calling processor if the calling processor were ceded, if there is no such pending work, the return code is H_Success. Due to race conditions, this pending work may have grown or disappeared by the time the calling OS makes a subsequent H_CEDE call.

There is **NO** architectural guarantee that ceding a processor exempts a virtual processor from preemption for a given period of time. That may indeed be the characteristic of a given implementation, but cannot be expected from all future implementations.

Syntax:

```
int64          /* H_Success      No pending platform work at this time */
               /* H_PENDING      There exists pending platform work */
               /* H_Hardware:    The hcall() experienced a hardware fault potentially preventing the function */
    hcall (const uint64 H_POLL_PENDING);          /* Poll for the presence of pending platform work */
```

Semantics:

- Return H_PENDING if there is work pending that could be dispatched to the calling processor if it were ceded, else return H_Success.

R1–14.11.3.9–1. For the SPLPAR option: The platform must implement the H_POLL_PENDING hcall() following the syntax and semantics of Section 14.11.3.9, “H_POLL_PENDING,” on page 465.

14.11.4 Pool Idle Count Function Set

The hcall-pic function set may be configured via the partition definition in none or any number of partitions as the weights administrative policy dictates.

14.11.4.1 H_PIC

Syntax:

```
int64          /* H_Success:     Expected return code */
               /* H_Function:    The function is not allowed from the calling partition */
               /* H_Authority:   The calling partition is not authorized to make the call at this time */
               /* H_Hardware:    The hcall() experienced a hardware fault potentially preventing the function */
    hcall (const uint64 H_PIC);          /*Returns in R4 the value of the Pool Idle Count */
```

Semantics:

- Verify that calling partition has the authority to make the call else return H_Authority.

- Compute the PIC value for the processor pool implementing the calling virtual processor up to the current point in time and place into R4
- Place the number of processors in the caller's processor pool in R5.
- When the value of the "ibm,partition-performance-parameters-level" (see Table B.6.2.1, "Root Node Properties," on page 673) is >=1 then:
  - Place the summation of time base ticks for all platform processors, allocated to the caller's processor pool, into register R6.
  - Place the summation of all PURR ticks accumulated by all dispatched (not idle) platform processor threads, allocated to the caller's processor pool, into register R7.
  - Place the summation of all SPURR[1] ticks accumulated by all dispatched (not idle) platform processor threads, allocated to the caller's processor pool, into register R8.
  - Place the caller's processor pool ID into low order two bytes of register R9 (high order 6 bytes are reserved - set to 0x000000).
  - If the calling partition has the authority to monitor total processor virtualization then:
    - Place the summation of time base ticks for all platform physical processors, allocated to processor virtualization, in register R10.
    - Place the summation of all PURR ticks accumulated by all dispatched (not idle) platform physical processor threads, allocated to processor virtualization, in register R11.
    - Place the summation of all SPURR[1] ticks accumulated by all dispatched (not idle) platform physical processor threads, allocated to processor virtualization, in register R12.
    - Else load R10, R11 and R12 with -1.
- Return H_Success.

R1–14.11.4.1–1. For the SPLPAR option: The platform must implement and make available to selected partitions, the H_PIC hcall() following the syntax and semantics of Section 14.11.4.1, "H_PIC," on page 465.

14.11.5 Thread Join Option

14.11.5.1 H_JOIN

The H_JOIN hcall() performs the equivalent of a H_CONFER (proc=self) hcall() (see Section 14.11.3.4, "H_CONFER," on page 460) unless called by the sole unjoined (active) processor thread, at which time the H_JOIN hcall() returns H_CONTINUE. H_JOIN is intended to establish a single threaded operating environment within a partition; to prevent external interrupts from complicating this environment, H_JOIN returns "bad_mode" if called with the processor MSR[EE] bit set to 1. Joined (inactive) threads are activated by H_PROD (see Section 14.11.3.5, "H_PROD," on page 462) which starts execution at the instruction following the hcall; or a system reset non-maskable interrupt which appears to interrupt between the hcall and the instruction following the hcall.

Syntax:

```
int64        /* H_Success     Return value to all processor threads in the calling partition except the final active
                               processor thread in the calling partition. */
```

1.Machines that do not have a SPURR mechanism are assumed to run at a constant speed, at which time the PURR value is substituted.

```
        /* bad_mode        MSR.EE=1 in addition to other illegal MSR bit values.
        /* H_CONTINUE Return value to final active processor thread in the calling partition. */
        /* H_Hardware      The hcall() experienced a hardware fault potentially preventing the function. */
hcall (const uint64 H_JOIN);/* Join active threads and return H_CONTINUE to final calling thread. */
```

Semantics:

- If MSR.EE=1 return bad_mode.
- If other processor threads are active in the calling partition, then emulate H_CONFER (proc=self)
- Else return H_CONTINUE.

R1–14.11.5.1–1. For the Thread Join option: The platform must implement the H_JOIN hcall() following the syntax and semantics of Section 14.11.5.1, “H_JOIN,” on page 466.

R1–14.11.5.1–2. For the Thread Join option: The platform must implement the hcall-join and hcall-splpar function sets.

R1–14.11.5.1–3. For the Thread Join option: The platform must support the H_PROD hcall even if the partition is operating in dedicated processor mode.

14.11.6 Virtual Processor Home Node Option (VPHN)

The SPLPAR option allows the platform to dispatch virtual processors on physical processors that due to the variable nature of work loads are temporarily free, thus improving the utilization of computing resources. However, SPLPAR implies inconsistent mapping of virtual to physical processors; defeating resource allocation software that attempts to optimize performance on platforms that implement the NUMA option.

To bridge the gap between these two options, the VPHN option maintain a substantially consistent mapping of a given virtual processor to a physical processor or set of processors within a given associativity domain. Thus the OS can, when allocating computing resources, take advantage of this statistically consistent mapping to improve processing performance.

VPHN mappings are substantially consistent but not static. For any given dispatch cycle, a best effort is made to dispatch the virtual processor on a physical processor within a targeted associativity domain (the virtual processor's home node). However, if processing capacity within the home node is not available, some other physical processor is assigned to meet the processing capacity entitlement. From time to time, to optimize the total platform performance, it may be necessary for the platform to change the home node of a given virtual processor.

To enable the OS to determine the associativity domain of the home node of a virtual processor, platforms implementing the VPHN option provide the H_HOME_NODE_ASSOCIATIVITY hcall(). The presence of the hcall-vphn function set in the **`“ibm,hypertas-functions”`** property indicates that the platform supports the VPHN option. The OS should be prepared for the support of the VPHN option to change with functions such partition migration, after which a call to H_HOME_NODE_ASSOCIATIVITY may end with a return code of H_FUNCTION. Additionally, the VPHN option defines a VPA field that the OS can poll to determine if the associativity domain of the home node has changed. When the home node associativity domain changes, the OS might choose to call the H_HOME_NODE_ASSOCIATIVITY hcall() and adjust its resource allocations accordingly.

R1–14.11.6–1. For the Virtual Processor Home Node option: The platform must support the H_HOME_NODE_ASSOCIATIVITY hcall() per the syntax and semantics specified in section 14.11.6.1, “H_HOME_NODE_ASSOCIATIVITY,” on page 468.

R1–14.11.6–2. For the Virtual Processor Home Node option: For the OS to operate properly across such functions as partition migration, the OS must be prepared for the target platform to not support the Virtual Processor Home Node option.

R1–14.11.6–3. For the Virtual Processor Home Node option: The platform must support the "virtual processor home node associativity changes counters" field in the VPA per section 14.11.6.2, "VPA Home Node Associativity Changes Counters," on page 469.

R1–14.11.6–4. For the Virtual Processor Home Node option: The platform must support the "Form 1" of the **"ibm,associativity-reference-points"** property per Section 15.3.2, "Form 1," on page 508. The client program may call H_HOME_NODE_ASSOCIATIVITY hcall() with a valid identifier input parameter (such as from the device tree or from the *ibm,configure-connector* RTAS call) even if the corresponding virtual processor has not been started so that the client program can allocate resources optimally with respect to the to be started virtual processor.

14.11.6.1 H_HOME_NODE_ASSOCIATIVITY

The H_HOME_NODE_ASSOCIATIVITY hcall() returns the associativity domain designation associated with the identifier input parameter. The client program may call H_HOME_NODE_ASSOCIATIVITY hcall() with a valid identifier input parameter (such as from the device tree or from the ***ibm,configure-connector*** RTAS call) even if the corresponding virtual processor has not been started so that the client program can allocate resources optimally with respect to the to be started virtual processor.

Syntax:

```
int64          /* H_Success:     Expected return code */
               /* H_Function:    The function is not supported (support may change following partition migration) */
               /* H_Parameter:   Unsupported flag parameter value */
               /* H_P2:          Invalid id parameter */
               /* H_Hardware:    The hcall() experienced a hardware fault potentially preventing the function */
    hcall (const uint64 H_HOME_NODE_ASSOCIATIVITY), /*Returns in R4-R9 the home node associativity */
                                                    /*domain IDs */
               uint64   flags,      /* Type of id parameter */
                                    /* 0x00 not used */
                                    /* 0x01 proc-no as used in the H_REGISTER_VPA hcall */
                                    /* 0x02 processor index as from byte offsets 0x2-0x3 of a trace log entry */
               uint64   id);        /* processor id in the form specified by the flags parameter*/
```

Parameters:

- Input:
  - flags:
    - NOTE: this parameter does not share format with the flags parameter of the Page Frame Table Access hcall()s.
    - Defined Values:
      - 0x0 Invalid
      - 0x1 id parameter is as proc-no parameter of H_REGISTER_VPA hcall()
      - 0x2 id parameter is as processor index from byte offsets 0x2-0x3 of a trace log buffer entry
      - all other values reserved.
  - id: processor identifier per the form indicated by the flags parameter.
- Output:
  - R3: return code

- R4-R9: associativity domain identifier list of the specified processor's home node.
  - Only the "primary" connection (as would be reported in the first string of the **"ibm,associativity"** property) is reported.
  - The associativity domain numbers are reported in the sequence they would appear in the **"ibm,associativity"** property; starting from the high order bytes of R4 proceeding toward the low order bytes of R9.
  - Each of the registers R4-R9 is divided into 4 fields each 2 bytes long.
  - The high order bit of each 2 byte field is a length specifier:
    - 1: The associativity domain number is contained in the low order 15 bits of the field,
    - 0: The associativity domain number is contained in the low order 15 bits of the current field concatenated with the 16 bits of the next sequential field)
  - All low order fields not required to specify the associativity domain identifier list contain the reserved value of all ones.

Semantics:

- Verify that the "flags" parameter is valid else return H_Parameter.
- Verify that the "id" parameter is valid for the "flags" and the partition else return H_P2.
- Pack the associativity domain identifiers for the home node associated with the "id" parameter starting with the highest level reported in the **"ibm,associativity"** property in the high order field of R4.
- All remaining fields through the low order field of R9 are filled with 0xFFFFFFFF.
- Return H_Success.

14.11.6.2 VPA Home Node Associativity Changes Counters

For the VPHN option, the platform maintains within each VPA the Virtual Processor Home Node Associativity Change Counters field. See Table 184, "Per Virtual Processor Area," on page 449. This eight (8) byte field is maintained as 8 one byte long counters. The number of counters that are supported is implementation dependent up to 8, and corresponds to the entries in the form 1 of the **"ibm,associativity-reference-points"** property. If the platform implements fewer than 8 associativity reference points, only the corresponding low offset counters within the field are used and the remaining high offset counters within the field are unused.

Should the associativity of the home node of the virtual processor change, for each changed associativity level that corresponds to a level reported in the **"ibm,associativity-reference-points"** property, the corresponding counter in the Virtual Processor Home Node Associativity Change Counters field is incremented.

14.12 Virtualizing Partition Memory

This section describes the various high level functions that are enabled by the virtualization of the logical real memory of a partition. In principle, virtualization of partition memory can be totally transparent to the partition software; however, partition software that is migration aware can cooperate with the platform to achieve higher performance, and enhanced functionality.

14.12.1 Partition Migration/Hibernation

Virtualizing partition memory allows a partition to be moved via migration or hibernation. In the case of partition migration from one platform to another, the source and destination platforms cooperate to minimize the time that the partition is non-responsive; the goal is to be non-responsive no more than a few seconds. In the case of hibernation, the intent is to put the partition to sleep for an extended period; during this time the partition state is stored on secondary storage for later restoration.

R1–14.12.1–1. For the Partition Migration and Partition Hibernation options: The platform must implement the Partition Suspension option (See Section 7.4.6, “ibm,suspend-me RTAS Call,” on page 243).

R1–14.12.1–2. For the Partition Migration and Partition Hibernation options: The platform must implement the VASI option (See Section 17.8, “Virtual Asynchronous Services Interface (VASI),” on page 715).

R1–14.12.1–3. For the Partition Migration and Partition Hibernation options: The platform must implement the Update OF Tree option.

R1–14.12.1–4. For the Partition Migration and Partition Hibernation options: The platform must implement the Version 6 Extensions of Event Log Format for all reported events (See Section 10.3.2.2, “Version 6 Extensions of Event Log Format,” on page 294).

R1–14.12.1–5. For the Partition Migration and Partition Hibernation options: The platform must prevent the migration/hibernation of partitions that own dedicated platform resources in addition to processors and memory, this includes physical I/O resources, the BSR facility, physical indicators and sensors (virtualized I/O, indicators (such as tone) and sensors (such as EPOW) are allowed).

R1–14.12.1–6. For the Partition Migration and Partition Hibernation options: The platform must implement the Client Vterm option.

R1–14.12.1–7. For the Partition Migration and Partition Hibernation options: The platform **`“time-base-frequency”`** must be 512 MHz. +/- 50 parts per million.

R1–14.12.1–8. For the Partition Migration and Partition Hibernation options: The platform must present the **`“ibm,nominal-tbf”`** property (See Section C.6.1.4, “CPU Node Properties,” on page 760) with the value of 512 MHz.

R1–14.12.1–9. For the Partition Suspension option: The platform must present the properties from Appendix C, “PA Processor Binding,” on page 753, as specified by Table 187, “Properties Related to the Partition Suspension Option,” on page 470, to a partition.

R1–14.12.1–10. For the Partition Suspension option: The presence and value of all properties in Table 187, “Properties Related to the Partition Suspension Option,” on page 470 must not change while a partition is suspended except for those properties described by Section 7.4.8, “ibm,update-properties RTAS Call,” on page 249.

Table 187. Properties Related to the Partition Suspension Option

Property Name	Requirement
`“ibm,estimate-precision”`	Shall be present. **`“ibm,estimate-precision”`** shall contain the “fre”, “fres”, frsqrte”, and “frsqrtes” instruction mnemonics.
`“ibm,processor-page-sizes”`	Shall be present.
`“reservation-granule-size”`	Shall be present.

Table 187. Properties Related to the Partition Suspension Option *(Continued)*

Property Name	Requirement
"cache-unified"	Shall be present if the cache is physically or logically unified and thus does not require the architected instruction sequence for data cache stores to appear in the instruction cache (See "Instruction Storage" section of Book II of PA); else shall not be present.
"i-cache-size"	Shall be present.
"d-cache-size"	Shall be present.
"i-cache-line-size"	Shall be present.
"d-cache-line-size"	Shall be present.
"i-cache-block-size"	Shall be present.
"d-cache-block-size"	Shall be present.
"i-cache-sets"	Shall be present.
"d-cache-sets"	Shall be present.
"timebase-frequency"	Shall be present if the timebase frequency can fit into the **"timebase-frequency"** property; else shall not be present.
"ibm,extended-timebase-frequency"	Shall be present if the timebase frequency cannot fit into the "**timebase-frequency"** property; else shall not be present.
"slb-size"	Shall be present.
"cpu-version"	Shall be present.
"ibm,ppc-interrupt-server#s"	Shall be present.
"l2-cache"	Shall be present if another level of cache exists; else shall not be present.
"ibm,vmx"	Shall be present if VMX is present for the partition; else shall not be present.
"clock-frequency"	Shall be present if the processor frequency can fit into the **"clock-frequency"** property; else shall not be present.
"ibm,extended-clock-frequency"	Shall be present if the processor frequency cannot fit into the "**clock-frequency"** property; else shall not be present.
"ibm,processor-storage-keys"	Shall be present.
"ibm,processor-vadd-size"	Shall be present.
"ibm,processor-segment-sizes"	Shall be present.
"ibm,segment-page-sizes"	Shall be present.
"64-bit"	Shall be present.
"ibm,dfp"	Shall be present if DFP is present for the partition; else shall not be present.

Table 187. Properties Related to the Partition Suspension Option ***(Continued)***

Property Name	Requirement
"ibm,purr"	Shall be present if a PURR is present; else shall not be present.
"performance-monitor"	Shall be present if a Performance Monitor is present; else shall not be present.
"32-64-bridge"	Shall be present.
"external-control"	Shall not be present.
"general-purpose"	Shall be present.
"graphics"	Shall be present.
"ibm,platform-hardware-notification"	Shall be present.
"603-translation"	Shall not be present.
"603-power-management"	Shall not be present.
"tlb-size"	Shall be present.
"tlb-sets"	Shall be present.
"tlb-split"	Shall be present.
"d-tlb-size"	Shall be present.
"d-tlb-sets"	Shall be present.
"i-tlb-size"	Shall be present.
"i-tlb-sets"	Shall be present.
"64-bit-virtual-address"	Shall not be present.
"bus-frequency"	Shall be present if the bus frequency can fit into the **"bus-frequency"** property; else shall not be present.
"ibm,extended-bus-frequency"	Shall be present if the processor frequency cannot fit into the "**bus-frequency"** property; else shall not be present.
"ibm,spurr"	Shall be present if an SPURR is present; else shall not be present.
"name"	Shall be present.
"device_type"	Shall be present.
"reg"	Shall be present.
"status"	Shall be present.
"ibm,pa-features"	Shall be present.
"ibm,negotiated-pa-features"	Shall be present
"ibm,ppc-interrupt-gserver#s"	Shall be present
"ibm,tbu40-offset"	Shall be present

Table 187. Properties Related to the Partition Suspension Option *(Continued)*

Property Name	Requirement
"ibm,pi-features"	Shall be present
"ibm,pa-optimizations"	Shall be present

Note on Table 187: The values of the properties in Table 187 shall be consistent with implementation and design of the processor and the platform upon boot as well as before and after partition suspension.

Programming Note: The **"cpu-version"** property may contain a logical processor version value. Therefore, code designed to handle processor errata should read the **"ibm,platform-hardware-notification"** property of the root node to obtain the physical processor version numbers allowed in the platform.

14.12.2 Virtualizing the Real Mode Area

PA requires implementations to provide a Real Mode Area of memory that is accessed when not in hypervisor state (either MSR[HV] = 0, or MSR[HV] = 1 and MSR[PR] = 1) and the OS address translation mechanism is disabled (MSR[IR] = 0 or MSR[DR] = 0). PA provides mechanisms to allow the RMA to consist of discontiguous pages of selectable sizes. Such an RMA is known as a virtualized RMA. The H_VRMASD hcall() allows the OS to change the characteristics of the mappings the address translation mechanism uses to access a virtualized RMA.

14.12.2.1 H_VRMASD

The caller may need to invoke the H_VRMASD hcall() multiple times for it to return with a return code of H_Success. Upon receiving a return code of H_LongBusyOrder10mSec, the caller should attempt to invoke H_VRMASD in 10 mSec with the same Page_Size_Code value used on the previous H_VRMASD hcall(). Invoking H_VRMASD with a different Page_Size_Code value indicates that the caller wants to transition to the Page_Size_Code value of the most recent H_VRMASD call.

When changing the page size used to map the VRMA using the H_VRMASD hcall(), the caller is responsible for establishing HPT entries for any potential real mode accesses prior to calling H_VRMASD with a new value of Page_Size_Code, and maintaining any HPT entries for the old value of Page_Size_Code until the hcall() returns H_Success.

R1–14.12.2.1–1. For the VRMA option: The platform must include the **"ibm,vrma-page-sizes"** property (See Appendix C, "PA Processor Binding," on page 753) in the **/cpu** node.

R1–14.12.2.1–2. For the VRMA option: The platform must implement the H_VRMASD hcall() following the syntax and semantics of Section 14.12.2.1, "H_VRMASD," on page 473.

R1–14.12.2.1–3. For the VRMA option: In order to prevent a storage exception, the calling partition must establish page table mappings for the Real Mode Area using entries with a page size corresponding to the new Page_Size_Code value prior to making an H_VRMASD hcall() and must maintain the old page table mappings using the page size corresponding to the old Page_Size_Code value until the H_VRMASD hcall() returns H_Success.

Syntax:

```
int64       /* H_Success,      Expected Return Code.*/
            /* H_LongBusyOrder10mSec          Retry calling the hcall() in 10 milliseconds.
            /* H_Parameter,    PAGE_SIZE_CODE parameter is invalid.*/
            /* H_Hardware,     The hcall() experienced a hardware fault preventing the function. */
```

```
hcall (const uint64 H_VRMASD,          /* Change the page mapping characteristics of the VRMA*/
       uint64 Page_Size_Code);         /* Contains a VRMASD field value */
```

Parameters:

- Page_Size_Code: A supported VRMASD field value. Supported VRMASD field values are described by the **`"ibm,vrma-page-sizes"`** property.

Semantics:

- Verify that the Page_Size_Code parameter corresponds to a supported VRMASD field value; else return H_Parameter.
- If the Real Mode Area page size specified by the Page_Size_Code parameter does not match the operating RMA page size of the partition, then set the operating RMA page size of the partition to the value specified by the Page_Size_Code parameter and initiate the transition of the operating RMA page size of all active processing threads to the value specified by the Page_Size_Code parameter.
- If all active threads have transitioned to the partition operating RMA page size, then return H_Success; else return H_LongBusyOrder10mSec.

14.12.3 Cooperative Memory Over-commitment Option (CMO)

The over-commitment of logical memory is accomplished by the platform reassigning pages of memory among the partitions to create the appearance of more memory than is actually present. This is commonly known as paging. While paging can, in certain cases, be accomplished transparently, significantly better memory utilization and platform performance can be achieved with cooperation from the partition OS.

CMO introduces the following LoPAPR terms:

- **Expropriation:** The act of the platform disassociating a physical page from a logical page.
- **Subvention:** The act of the platform associating a physical page with a logical page.
- **Loaned Memory:** Logical real memory that a partition lends to the hypervisor for reuse. The partition should not gratuitously access loaned memory as such accesses are likely to experience a significant delay.
- **Memory entitlement**: The amount of memory that the platform guarantees that the partition is able to I/O map at any given time.

R1–14.12.3–1. For the CMO option: The partition must be running under the SPLPAR option.

R1–14.12.3–2. For the CMO option: The platform must transparently (except for time delays) handle all effects of any memory expropriation that it may introduce unless the CMO option is explicitly enabled by the setting of architecture.vec option vector 5 byte 4 bit 0 (See Table 244, “ibm,architecture.vec option vectors,” on page 681 for details).

The CMO option consists of the following LoPAPR extensions:

- Define **`ibm,architectue.vec-5`** option Byte 4 bit 0 as “Client supports cooperative logical real memory over-commitment”.
- Define page usage states to assist the platform in selecting good victim pages and mechanisms to set such states.
- Extend the syntax and semantics defined for the I/O mapping hcall()s
  - Return codes (H_LongBusyOrder1msec, H_LongBusyOrder10msec, and H_NOT_ENOUGH_RESOURCES)
  - Return parameter extension for memory entitlement management

- Define a simulated Special Uncorrectable memory Error machine check for the case where a page can not be restored due to an error.

R1–14.12.3–3. For the CMO option: The architected interface syntax and semantics of all LoPAPR hcall()s and RTAS calls except as explicitly modified per the CMO option architecture must remain invariant when operating in CMO mode; any accommodation to memory over-commitment by these firmware functions (potentially any function that takes a logical real address as an input parameter) is handled transparently.

NOTE: Requirement R1–14.12.3–3 specifically applies to the debugger support hcall()s.

For maximum performance benefit, an OS that indicates via the **`ibm,client-architecture-support`** interface that it supports the CMO option will strive to maintain in the "loaned" state (See Section 14.12.3.2, "CMO Page Usage States," on page 477), the amount of logical memory indicated by the value returned in R9 from the H_GET_MPP hcall (See Section 14.12.3.5, "H_GET_MPP," on page 482), as well as provide page usage state information via the interfaces defined in Section 14.12.3.2.1, "Setting CMO Page Usage States using HPT hcall() flags Parameter," on page 478 and Section 14.12.3.2.2, "Setting CMO Page Usage States with H_BULK_REMOVE," on page 479.

The Extended Cooperative Memory Over-commitment Option (XCMO) provides additional features to manage page coalescing. These features are activated via setting architecture.vec vector 5 byte 4 bit 1 to the value of 1 in the **`ibm,client-architecture-support`** interface. Given that the platform supports the XCMO option, the CC flag for page frame table Accesses see Table 178, "Page Frame Table Access flags field definition," on page 401and the H_GET_MPP_X hcall() see Section 14.12.3.5.1, "H_GET_MPP_X," on page 483 may be used by the OS**.** An OS might understand that a given page is a great candidate for page coalescing perhaps because the page contains OS and or common library code which is likely to be duplicated in other partitions; if so it might choose to set the Coalesce Candidate (CC) flag in the page table access or H_PAGE_INIT hcall()s as a hint to the hypervisor. Should a given logical page be mapped multiple times with conflicting Coalesce Candidate hints, the value in the last mapping made takes precedence.

For a variety of reasons outside the scope of LoPAPR, a platform supporting the XCMO option for a given platform might not actually perform page coalescing. If this is the case, the first return value from the H_GET_MPP_X hcall() see Section 14.12.3.5.1, "H_GET_MPP_X," on page 483 is the reserved value zero.

R1–14.12.3–4. For the XCMO Option: The platform must implement the CMO Option.

R1–14.12.3–5. Reserved for Compatibility For the XCMO Option: The platform must implement the CC (Coalesce Candidate) flag bit see Table 178, "Page Frame Table Access flags field definition," on page 401.

R1–14.12.3–6. For the XCMO Option: The platform must implement the H_GET_MPP_X hcall() see Section 14.12.3.5.1, "H_GET_MPP_X," on page 483.

R1–14.12.3–7. For the XCMO Option with the Partition Migration and Partition Hibernation options: to ensure proper operation after partition migration or hibernation, the OS must stop setting the CC flag bit see Table 178, "Page Frame Table Access flags field definition," on page 401 and stop calling the H_GET_MPP_X hcall() see Section 14.12.3.5.1, "H_GET_MPP_X," on page 483 prior to calling *ibm,suspend-me* RTAS and not do so again until after the OS has determined that the XCMO option is supported on the destination platform.

14.12.3.1 CMO Background (Informative)

The following information is provided to be informative of the architectural intent. Implementations may vary, but should make a best effort to achieve the goals described.

Ideally, the hypervisor does not expropriate any logical memory pages that it must later read in from disk; this is based upon the belief that the OS is in a better position to determine its working set relative to the available memory and page its memory than the hypervisor thus, when possible, the OS pager should be used. The ideal is approximated, since it

cannot be achieved in all cases. The “Overage” is defined as the amount of logical address space that cannot be backed by the physical main storage pool. The overage is equal to the summation of the logical address space for all partitions using a given VRM main storage pool (the main storage that the hypervisor uses to back logical memory pages for a set of partitions) plus the high water mark of the hypervisor free page list (the free list high water mark is some implementation dependent ratio of the pool size) less the size of the VRM main storage pool.

If the summation of the space freed by page coalescing and page donation is equal to the overage, in the steady state the hypervisor need not page. In reality the system is seldom, if ever, in the steady state, but with the free list pages the hypervisor has enough buffer space to take up most of the transient cases.

Page coalescing is a transparent operation where in the hypervisor detects duplicate pages, directs all user reads to a single copy and may reclaim the other duplicate physical memory pages. Should the page owner change a coalesced page the hypervisor needs to transparently provide the page owner with a unique copy of the page. Read only pages are more likely to remain identical for a longer period of time and are thus better coalescing candidates.

To set the value for the partition's page donation, the algorithm needs to be “fair” and responsive to the partition's “weight” so that more important work can be helped along. To be “fair”, the donation needs to be somewhat proportional to the partition's size since donating x pages is likely to cause greater pain to a small partition than a large one; yet the reason for “weight” is to cause greater pain to certain partitions relative to others.

Thus the initial donation for a partition is set at the partition's logical address space size as a percentage of the total pool logical space subscription times the overage.

Each implementation dependent time interval (say single digit seconds or so), the hypervisor randomly selects 100 pages from each partition and monitors how many of them were accessed during the next interval. This, after normalization to account for partition CPU utilization relative to its recent maximum, becomes an estimate of the partition's page utilization. It is expected that a partition with higher page utilization has a higher page fault rate and a lower percentage of its working set resident -- thus experiences more pain from VRM.

The page utilization method described above may over estimate memory pressure in certain cases; specifically it may be slow to realize that the partition has gone idle. An idle partition reduces its CPU utilization which after normalization makes it appear that the partition memory pressure has risen rather than lowered. For this reason, the results of the page utilization method is further compared with the OS reported count of faults against pages that were previously swapped out as reported in offsets 0x180 – 0x183 of the VPA for each of the partition processors. The partition fault count when normalized with respect to processor cycles allows comparisons among the reported values from other partitions. Since the partition fault count is OS reported, and thus can not be trusted, it can not be the primary value used to determine page allocation, but since if the OS is misreporting the statistic, it is likely to be high, the memory pressure estimate derived from the OS reported fault counts can be used to reduce (but not increase) the partition memory allocation. Note since the hint might not be reported by a given OS, a filter should be put in place to detect that the OS is not reporting faults and appropriate default values substituted.

This initial donation is then modified over time to force the pain of higher page utilization upon lower weight partitions based upon comparing the following ratios:

A: The average partition page utilization over the last interval of all partitions in the pool / the partition's page utilization over the last interval

B: The partition's weight/average partition weight of all partitions in the pool

If A > B Increase the partition's donation by 1/256 of the partition's logical address space (limited to the partition’s logical address size)

If A < B Reduce the partition's donation by 1/256 of the partition's logical address space provided that the summation of all donations >= Overage.

The hypervisor maintains a per partition count of loaned pages (incremented when a page is removed from the PFT with a “loaned” state and decremented when/if the page state is changed) thus it can keep track of how well a partition is doing against the donation request that has been made of it. Partitions that do not respond to donation requests need

to have their pages stolen to make up the difference. Pages that are “unused” or “loaned” are automatically applied to the free list. “Loaned” pages are expected to raise the partition's free list low water mark so that the OS only reclaims them in a transient situation which will then result in the OS paging out some of its own virtual memory to restore the total donation in the steady state. When the platform free list gets to the low water mark, pages are expropriated starting with the partition that has the greatest percentage discrepancy between its loaned plus expropriated count and is donation tax. The algorithm used is implementation dependent. The following is given for reference and is loosely based upon the AIX method.

1. For this algorithm, pages that are newly restored are marked as “referenced” and all “unused” have already been harvested
2. Step through the partition logical address space until either the hypervisor free list has gotten to its high water mark or the partition has been taxed to its donation.
    a. If the page is I/O mapped and not expropriatable, continue to the next page.
    b. If this is the first pass through the address space on this harvest, and the page is marked critical, continue to the next page.
    c. If the page is marked “referenced”, clear the reference bit and continue to the next page.
    d. If the page is backed in the VPM paging space and not modified since then, expropriate the page and continue to the next page.
    e. Queue the page to be copied into the VPM paging space.

Thus partitions that keep up with their page donations seldom, if ever, experience a hypervisor page in delay. Those that do not keep up, will not get a free lunch and will be taxed up to the value of their assigned donation, with the real possibility that they will experience the pain of hypervisor page in delays.

14.12.3.2 CMO Page Usage States

The CMO option defines a number of page states that are set by the cooperating OS using the flags parameter of the HPT hcall()s. The platform uses these page states to estimate the overhead associated with expropriating the specific page.

Note: that the first two definitions below represent base background page states; the 3rd definition is the foreground state of I/O mapped which is acquired as result of an I/O mapping hcall (such as H_PUT_TCE); and the last two are caller specifiable state modifiers/extended semantics of the base states.

- **Unused –** the page contains no information that is needed in the future, its contents need not be maintained by the platform, normally set only when the page is unmapped.[1]
- **Active –** the page retains data that the OS has no reasonable way to regenerate. This is the state traditionally assumed by the OS when mapping a page.[2]
- **I/O Mapped –** the page is mapped for access by another agent. This state is the side effect of registration and/or I/O mapping functions. The page returns to its background state automatically when unmapped or deregistered.[3]

1.Expropriation of “Unused” pages should be a low overhead operation. However, the OS is likely to reuse these pages which means that a clean free page will have to be assigned to the corresponding logical address.

2.“Active” pages should be expropriated only as a last resort since they must be paged out and paged back in on a subsequent access.

- **Critical –** the page is critical to the performance of the OS, and the hypervisor should avoid expropriating such pages while other pages are available.[1]
- **Loaned –** the page contains no information and the OS warrants that it will not gratuitously access this page such that the hypervisor may expect to use it for an extended period of time. When the OS does access the page, it is likely that the access will result in a subvention delay.[2]

R1–14.12.3.2–1. For the CMO option: The platform must at partition boot initialize the page usage state of all platform pages to “Active”.

R1–14.12.3.2–2. For the CMO option: The platform must preserve data in pages that are in the “Active” state.

R1–14.12.3.2–3. For the CMO option: When the OS accesses a page in the “Unused” state, the platform must present either the preserved page data or all zeros.

R1–14.12.3.2–4. For the CMO option: When the OS specifies as input to an I/O mapping or the H_MIGRATE_DMA hcall() a page in either the “Unused” or “Loaned” states, the platform must upgrade the page’s background page state to “Active”.

14.12.3.2.1 Setting CMO Page Usage States using HPT hcall() flags Parameter

The CMO option defines additional flags parameter combinations for the HPT hcall()s that take a flags parameter. Turning on flags bit 28 activates the changing of page state. Leaving bit 28 at the legacy value of zero maintains the page state setting, thus allowing legacy code to operate unmodified with all pages remaining in the initialized “Active” state.

R1–14.12.3.2.1–1. For the CMO option: The platform must extend the syntax and semantics of the HPT access hcall()s that take a flags parameter, see Table 188, “HPT hcall()s extended with CMO flags,” on page 478, to set the page usage state of the specified page per Table 189, “CMO Page Usage State flags Definition,” on page 479.

Table 188. HPT hcall()s extended with CMO flags

hcall
Section 14.5.4.1.1, “H_REMOVE,” on page 408
Section 14.5.4.1.2, “H_ENTER,” on page 410
Section 14.5.4.1.3, “H_READ,” on page 414
Section 14.5.4.1.4, “H_CLEAR_MOD,” on page 414
Section 14.5.4.1.5, “H_CLEAR_REF,” on page 415
Section 14.5.4.1.6, “H_PROTECT,” on page 416
Section 14.5.4.3.3, “H_PAGE_INIT,” on page 424

3.Pages in the I/O Mapped state normally may not be expropriated since they are potentially the target of physical DMA operations.

1.Expropriating pages marked “Critical” may result in the OS being unable to meet its performance goals.

2.Expropriating pages in the “Loaned” state should result in the lowest overhead.

Table 189. CMO Page Usage State flags Definition

Flag bit 28	Flag bits 29 - 30	Flag bit 31	Comments
0	Don't Care	Don't Care	Inhibit Page State Change
1	00	0	Set page state to Active
	00	1	Set page State to Active Critical
	01	both 0 and 1	Reserved
	10	both 0 and 1	Reserved
	11	0	Set page state to Unused
	11	1	Set page state to Unused Loaned

14.12.3.2.2 Setting CMO Page Usage States with H_BULK_REMOVE

R1–14.12.3.2.2–1. For the CMO option: The platform must extend the syntax and semantics of the H_BULK_REMOVE hcall (see 14.5.4.1.7, "H_BULK_REMOVE," on page 417) to set the page usage state of the specified pages per Table 190, "H_BULK_REMOVE Translation Specifier control/status Byte Extended Definition for CMO Option," on page 480.

Table 190. H_BULK_REMOVE Translation Specifier control/status Byte Extended Definition for CMO Option

0	1	2	3	4	5	6	7	Bit Numbers
type code								
0	0	r0	r0	r0	r0	r0	r0	Unused
0	1	**page state**		r0	r0	req. mod.		Request
						0	0	Absolute
						0	1	andcon
						1	0	APVN
						1	1	not used
		0	**0**	**Inhibit page usage state change**				
		0	**1**	**Reserved**				
		1	**0**	**For CMO option set page usage state to “Unused” if Success**				
		1	**1**	**For CMO option set page usage state to “Loaned” if Success**				
1	0	return code		Response				
		0	0	R	C	r	r	Success
		0	1	r	r			Not Found
		1	0					H_PARM
		1	1					H_HW
1	1	Reserved (to be zero)						End of String

Legion R=Reference Bit, C=Change Bit, r=reserved ignore, r0=reserved to be zero

14.12.3.3 CMO Extensions for I/O Mapping Hcall()s

If an OS were to map an excessive amount of its memory for potential physical DMA access, little of its memory would be left for paging; conversely, if the OS was totally prevented from I/O mapping its memory, it could not do I/O operations. The CMO option introduces the concept of memory entitlement. The partition’s memory entitlement is the amount of memory that the platform guarantees that the partition is able to I/O map at any given time. A given page may be mapped multiple times through different LIOBNs yet it only counts once against the partition’s I/O mapping memory entitlement. The syntax of certain I/O mapping hcall()s is extended to return the change in the partition’s I/O mapped memory total. The entitlement is intended to be used to ensure forward progress of I/O operations.

R1–14.12.3.3–1. For the CMO option: When the partition is operating in CMO mode, the platform must extend the syntax and semantics of the I/O mapping hcall()s specified in Table 191, “I/O Mapping hcall()s Modified by the CMO Option.,” on page 481 as per the specifications in Section 14.12.3.3.1, “CMO I/O Mapping Ex-

tended Return Codes," on page 481 and Section 14.12.3.3.2, "CMO I/O Mapping Extended Return Parameter," on page 481.

Table 191. I/O Mapping hcall()s Modified by the CMO Option.

hcall()	Base Definition on
H_PUT_TCE	H_PUT_TCE / 14.5.4.2.2
H_STUFF_TCE	H_STUFF_TCE / 14.5.4.2.3
H_PUT_TCE_INDIRECT	H_PUT_TCE_INDIRECT / 14.5.4.2.4

NOTE: The I/O mapping hcalls H_PUT_RTCE and H_PUT_RTCE_INDIRECT do not change the number of pages that are I/O mapped since they simply create copies of the I/O mappings that already exist.

14.12.3.3.1 CMO I/O Mapping Extended Return Codes

R1–14.12.3.3.1–1. For the CMO option: The platform must ensure that the DMA agent operating through the I/O mappings established by the hcall()s specified in Table 191, "I/O Mapping hcall()s Modified by the CMO Option.," on page 481 can appear to successfully access the associated page data of any expropriated page referenced by the input parameters of the hcall() prior to returning the code H_Success.

R1–14.12.3.3.1–2. For the CMO option: The platform must either extend the return code set for the hcall()s specified in Table 191, "I/O Mapping hcall()s Modified by the CMO Option.," on page 481 to include H_LongBusyOrder1msec and/or H_LongBusyOrder10msec or transparently suspend the calling virtual processor for cases where the function is delayed pending the restoration of an expropriated page.

R1–14.12.3.3.1–3. For the CMO option: The platform must extend the return code set for the hcall()s specified in Table 191, "I/O Mapping hcall()s Modified by the CMO Option.," on page 481 to include H_NOT_ENOUGH_RESOURCES for cases where the function would cause more memory to be I/O mapped than the caller is entitled to I/O map and the platform is incapable of honoring the request.

14.12.3.3.2 CMO I/O Mapping Extended Return Parameter

The syntax and semantics of the hcall()s in Table 191, "I/O Mapping hcall()s Modified by the CMO Option.," on page 481 are extended when the partition is operating in CMO mode by returning in register R4 the change in the partition's total number of I/O mapped memory bytes due to the execution of the hcall(). The number may be positive (increase in the amount of memory mapped) negative or zero (the page was/remains mapped for I/O access by another agent).

R1–14.12.3.3.2–1. For the CMO option: The platform must extend the syntax and semantics of the hcall()s specified in Table 191, "I/O Mapping hcall()s Modified by the CMO Option.," on page 481 when operating in CMO mode, to return in register R4 the change to the total number of bytes that were I/O mapped due to the hcall().

14.12.3.4 H_SET_MPP

This hcall() sets, within limits, the partition's memory performance parameters. If the request parameter exceeds the constraint of the calling LPAR's environment, the hypervisor limits the value set to the constrained value and returns H_Constrained. The memory weight is architecturally constrained to be within the range of 0-255.

Syntax:

```
int64          /* H_Success:      Expected return code */
               /* H_Constrained: The input parameter exceeds the partition's constraints*/
               /* H_Hardware:    The hcall() experienced a hardware fault potentially preventing the function */
    hcall (const uint64 H_SET_MPP),    /*Sets the Memory Performance Parameters within constraints. */
               uint64   entitled,      /* I/O mapping memory entitlement in bytes */
               uint8    variable;      /* The memory weight used to determine the page victim partition */
```

Semantics:

- Verify that the memory performance parameters specified are within the constraints of the partition:
  - If yes, atomically set the partition's memory performance parameters per the request and return H_Success.
  - If not, set the partition's memory performance parameters as constrained by the partition's configuration and return H_Constrained.

R1–14.12.3.4–1. For the CMO option: The platform must initially set the partition memory performance parameters to their configured maximums at partition boot time.

R1–14.12.3.4–2. For the CMO option: The platform must implement the H_SET_MPP hcall() following the syntax and semantics of Section 14.12.3.4, "H_SET_MPP," on page 481.

R1–14.12.3.4–3. For the CMO option: The platform must constrain the partition memory weight to the range 0-255.

14.12.3.5 H_GET_MPP

This hcall() reports the partition's memory performance parameters. The returned parameters are packed into registers.

Syntax:

```
int64          /* H_Success:      Expected return code */
               /* H_Hardware:    The hcall() experienced a hardware fault potentially preventing the function */
    hcall (const uint64 H_GET_MPP);    /*Returns in R4 - R10 the Memory Performance Parameters. */
```

Semantics:

- Place the partition's memory performance parameters for the calling virtual processor's partition into the respective registers:
  - R4: The number of bytes of main storage that the calling partition is entitled to I/O map. In the case of a dedicated memory partition this shall be the size of the partition's logical address space.
  - R5: The number of bytes of main storage that the calling partition has I/O mapped. In the case of a dedicated memory partition this is not applicable which is represented by the code -1.
  - R6: The calling partition's virtual partition memory aggregation identifier numbers, up to 4 levels:
    - Bytes 0-1: Reserved for future aggregation definition, and set to zero -- in the future this field may be given meaning.
    - Bytes 2-3: Reserved for future aggregation definition, and set to zero -- in the future this field may be given meaning.
    - Bytes 4-5: 16 bit binary representation of the "Group Number".

- Bytes 6-7: 16 bit binary representation of the “Pool Number”. In the case of a dedicated memory partition the “Pool Number” is not applicable which is represented by the code 0xFFFF.

- R7: Collection of short memory performance parameters for the calling partition:
  - Byte 0: Memory weight (0-255). In the case of a dedicated processor partition this is not applicable which is represented by the code 0.
  - Byte 1: Unallocated memory weight for the calling partition’s aggregation.
  - Bytes 2-7: Unallocated I/O mapping entitlement for the calling partition’s aggregation divided by 4096.
- R8: The calling partition’s memory pool main storage size in bytes. In the case of a dedicated processor partition this is not applicable which is represented by the code -1.
- R9: The signed difference between the number of bytes of logical storage that are currently on loan from the calling partition and the partition’s overage allotment (a positive number indicates a request to the partition to loan the indicated number of bytes else they will be expropriated as needed).
- R10: The number of bytes of main storage that is backing the partition logical address space. In the case of a dedicated processor partition this is the size of the partition’s logical address space.
- Return H_Success.

R1–14.12.3.5–1. For the CMO option: The platform must implement the H_GET_MPP hcall() following the syntax and semantics of Section 14.12.3.5, “H_GET_MPP,” on page 482.

14.12.3.5.1 H_GET_MPP_X

This hcall() provides additional information over and above (not duplication of) that which is returned by the H_GET_MPP hcall() See Section 14.12.3.5, “H_GET_MPP,” on page 482. The syntax of this hcall() is specifically designed to be seamlessly extensible and version to version compatible both from the view of the caller and the called on an invocation by invocation basis. To this end, all return registers (R3 (return code) through R10) are defined from the outset, some are defined as reserved and are set to zero upon return by the hcall(). The caller is explicitly prohibited from assuming that any reserved register contains the value zero, so that there will be no incompatibility with future versions of the hcall() that return non-zero values in those registers. New definitions for returned values will define the value zero to indicate a benign or unreported setting.

Syntax:

```
Int64                          /* H_Success: Expected return code */
                               /* H_Hardware The hcall() experienced a hardware fault potentially preventing the */
                               /* function */
                               /* H_Function The platform does not implement the hcall() */
  Hcall  (const H_GET_MPP_X);        /* Returns in R4-R10 extended Memory Performance/*
                                     /* Parameters */
```

Semantics:

- Place the partition’s extended memory performance parameters for the calling virtual processor’s partition into the respective registers:
  - R4: The number of bytes of the calling partition’s logical real memory coalesced because they contained duplicated data.
  - R5: If the calling partition is authorized to see pool wide statistics (set by means that are beyond the scope of LoPAPR) then The number of bytes of logical real memory coalesced because they contained duplicated data in the calling partition’s memory pool else set to zero.

- R6:: If the calling partition is authorized to see pool wide statistics (set by means that are beyond the scope of LoPAPR) then PURR cycles consumed to coalesce data else set to zero.
- R7: If the calling partition is authorized to see pool wide statistics (set by means that are beyond the scope of LoPAPR) then SPURR cycles consumed to coalesce data else set to zero.
- R8: Reserved shall be set to zero – shall not be read by the caller
- R9: Reserved shall be set to zero – shall not be read by the caller
- R10: Reserved shall be set to zero – shall not be read by the caller

◆ Return H_Success:

R1–14.12.3.5.1–1. For the XCMO option: If the platform coalesces memory pages that contain duplicated data it must implement the H_GET_MPP_X hcall() following the syntax and semantics of Section 14.12.3.5.1, “H_GET_MPP_X,” on page 483.

R1–14.12.3.5.1–2. For the XCMO option: the caller must be prepared for H_GET_MPP_X to return H_Function or to have a return parameter that was previously non-zero be consistently returned with the value zero if the caller wishes to operate properly in a partition migration or fail-over environment.

14.12.3.6 Restoration Failure Interrupt

R1–14.12.3.6–1. For the CMO option: When the platform experiences an unrecoverable error restoring the association of a physical page with an expropriated logical page following an attempted access of the expropriated page by the partition, the platform must signal a Machine Check Interrupt by returning to the partition’s interrupt vector at location 0x0200. Note the subsequent firmware assisted NMI and check exception processing returns a VPM SUE error log (See Section 10.3.2.2.12, “Platform Event Log Format, Failing Memory Address,” on page 311).

14.12.3.7 H_MO_PERF

This hcall() applies an artificial memory over-commitment to the specified pool while monitoring the pool performance for overload, removing the applied over-commitment if an overload trigger point is reached. The overload trigger point is designed to double as a dead man switch, eventually ending the over-commitment condition should the experiment terminate ungracefully. Only the partition that is authorized to run platform diagnostics is authorized to make this call.

Syntax:

```
int64       /* H_Success:      Expected return code */
            /* H_Hardware:     The hcall() experienced a hardware fault potentially preventing the function */
            /* H_AUTHORITY:          The caller lacked the authority to make this call*/
            /* H_Parameter:    The Pool number parameter is invalid */
    hcall (const uint64 H_MO_PERF),    /*Returns the accumulated pool memory and performance load*/
            uint16   pool,     /* Pool number being loaded */
            int      mem,      /* The change to the accumulated “bench” from the memory pool */
            int      lows);    /* The change to the accumulated permissible pool memory low events */
```

Semantics:

This description is based upon the architectural model of 14.12.3.1, “CMO Background (Informative),” on page 475, and must be adjusted to achieve the intent for the specific implementation.

◆ Validate that the caller has the required authority; else return H_AUTHORITY.

- Validate that the pool parameter references an active memory pool else; return H_Parameter.
- Raise the pool's free list low water mark above its base value by the signed amount in the mem parameter. (The result is constrained to not less than the base low water mark value and no more then the amount of memory in the pool.)
- Change the permissible pool memory low event counter by the signed value of the lows parameter.
- Return in R4 the accumulated rise in the pool's free list low water mark above its base value.
- Return in R5 the current value of the permissible pool memory low event counter.
- On each subsequent low memory event (page allocation where the free list is at or below the low water mark), the permissible pool memory low event counter is decremented. Should the counter ever reach zero, the pool's free list low water mark is returned to its base value.

14.12.3.8 Expropriation/Subvention Notification Option

The Expropriation/Subvention Notification Option (ESN) sub option of the CMO option allows implementing platforms to notify supporting OS's of delays due to their access of an expropriated VPM page (such as would be experienced during a "page in" operation). With an expropriation notification, the OS may block the affected process and dispatch another rather than having the platform block the virtual processor that happened to be running the affected process. An expropriation notification is paired with a subsequent subvention notification signaled when the original access succeeds. Additionally new page states allow the OS to indicate pages that it can restore itself, thus relieving the platform from the burden of making copies of those pages when they are expropriated and potentially side stepping the "double paging problem" wherein the platform pages in a page in response to a touch operation in preparation for an OS page in only to have the OS immediately discard the page data without looking at it.

The ESN option includes the following LoPAPR extensions:

- Define augmented CMO page states
- Define the per partition Subvention Notification Structure (SNS)
- Define H_REG_SNS hcall() to register the SNS
- Define Expropriation Notification field definitions within the VPA
- Define expropriation and subvention event interrupts.

R1–14.12.3.8–1. For the ESN option: The partition must be running under the CMO option.

R1–14.12.3.8–2. For the ESN option: The platform must ignore/disable all other ESN option functions and features unless the OS has successfully registered the Subvention Notification Structure via the H_REG_SNS hcall. See Section 14.12.3.8.3.2, "SNS Registration (H_REG_SNS)," on page 489 for details.

14.12.3.8.1 ESN Augmentation of CMO Page Usage States

The ESN option augments the set of page states defined by the CMO option that are set by the cooperating OS using the flags parameter of the HPT hcall()s. The platform uses these page states to estimate the overhead associated with expropriating the specific page.

Active – An Expropriation notification on this type of page allows the OS to put the using process to sleep, until the page is restored, as signaled by a corresponding subvention notification, at which time the affected instruction is retried.

Expendable – the page retains data that the OS can regenerate, for example, a text page that is backed up on disk; usually the page is mapped read only. A reflected expropriation notification on this type of page re-

quires the OS to restore the page – thus the platform presents somewhat different interrupt status from that used by an Active page.[1] [2]

Latent – the page contains data that the OS can regenerate unless the contents have been modified – at which time the page state appears to be “Active”, this is similar to “Expendable” for pages mapped read/write. For example, a page of a mapped file. Expropriation Notification is like “Active” or “Expendable” above.

Loaned – When the OS does access the page, it is likely that the access will result in an expropriation notification

Table 192. ESN Augmentation of CMO Page Usage State flags Definition

Flag bit 28	Flag bits 29 - 30	Flag bit 31	Comments
1	01	0	Set page state to Latent[a]
	01	1	Set page State to Latent Critical[a]
	10	0	Set page state to Expendable[a]
	10	1	Set page state to Expendable Critical[a]

a. NOTE: If Expropriation Notification is disabled, or the bolted bit (HPT bit 59) is set to 1, the page state to Active (Active Critical if flag bit 31=1).

14.12.3.8.2 Expropriation Notification

Under the ESN option, notice of an attempt to access an expropriated page is given when the Expropriation Interrupt is enabled in the virtual processor VPA. Additionally the virtual processor VPA Expropriation Correlation Number and Expropriation Flags fields are set to allow the affected program to determine when the access may succeed and if the program needs to restore data to the Subvened page, see details in Section 14.12.3.8.2.1, “ESN VPA Fields,” on page 486. Once the VPA has been updated, the platform presents an Expropriation Fault interrupt to the affected virtual processor see details in Section 14.12.3.8.2.2, “Expropriation Interrupt,” on page 487.

14.12.3.8.2.1 ESN VPA Fields

R1–14.12.3.8.2.1–1. For the ESN option: The platform must support the VPA field definitions of Table 193, “VPA Byte Offset 0xB9,” on page 486, Table 194, “Firmware Written VPA Starting at Byte Offset 0x178,” on page 487, and Table 195, “Expropriation Flags at VPA Byte Offset 0x17D,” on page 487.

Table 193. VPA Byte Offset 0xB9

0	1	2	3	4	5	6	7	Bit Number
Reserved (0)							0	Dedicated processor cycle donation inhibited
							1	Dedicated processor cycle donation enabled
						0		Expropriation interrupt disabled
						1		Expropriation interrupt enabled

1.Expropriating an “Expendable” page should result in lower overhead than expropriating an “Active” page since the contents need not be paged out before the page is reused.

2.An Expendable page that is Bolted while not illegal has to be treated as an “Active” page since an access to a Bolted page may not result in an expropriation notification.

Table 194. Firmware Written VPA Starting at Byte Offset 0x178

0x178 F	0x179	0x17A	0x17B	0x17C	0x17D	0x17E	0x17
Reserved for firmware locks	Reserved					Expropriation Correlation Number Field	
					Expropriation Flags -- See Table 195, “Expropriation Flags at VPA Byte Offset 0x17D,” on page 487.		

Note: The Expropriation Flags and Expropriation Correlation Number Fields are volatile with respect to Expropriation Notifications thus it should be saved by the OS before executing any instruction that may access unbolted pages.

Table 195. Expropriation Flags at VPA Byte Offset 0x17D

0	1	2	3	4	5	6	7	Bit Number
Reserved (0)							0	The Subvened page data is/will be zero
							1	The Subvened page data will be restored.

14.12.3.8.2.2 Expropriation Interrupt

When the platform is running with real memory over-commitment, eventually a partition virtual processor will access a stolen page. The transparent solution is to block the virtual processor until the platform has restored the page. By enabling the Expropriation Interrupt via the Expropriation Interrupt Enable field of the VPA (see Section 14.12.3.8.2.1, “ESN VPA Fields,” on page 486) the cooperating OS indicates that it is prepared to make use of its virtual processors for other purposes during the page restoration and/or restore the contents of “expendable” and unmodified “latent” pages.

R1–14.12.3.8.2.2–1. For the ESN option: When the partition accesses an expropriated page and either the page was bolted (PTE bit 59=1) or the Expropriation Interrupt Enable bit of the affected virtual processor’s VPA is off see Section 14.12.3.8.2.1, “ESN VPA Fields,” on page 486, then the platform must recover the page transparently without an Expropriation Interrupt.

R1–14.12.3.8.2.2–2. For the ESN option: When the partition accesses an expropriated page and the summation of the partition’s in use subvention event queue entries plus outstanding subvention events is equal to or greater than the size of the partition’s subvention event queue, the platform must recover the page prior to issuing any associated Expropriation Interrupt.

Note: Requirement R1–14.12.3.8.2.2–2 prevents the overflow of the subvention queue.

R1–14.12.3.8.2.2–3. For the ESN option: When the partition accesses an expropriated “Unused” or “Expendable” page, the platform must, unless prevented by R1–14.12.3.8.2.1–1, set bit 7 of the affected processor’s Expropriation Flags VPA byte (see Table 195, “Expropriation Flags at VPA Byte Offset 0x17D,” on page 487) to 0b0; else the platform must set the bit to 0b1.

R1–14.12.3.8.2.2–4. For the ESN option: When the partition accesses an expropriated page and the platform associates a physical page with the logical page prior to returning control to the affected virtual processor, the platform must, unless prevented by R1–14.12.3.8.2.1–1, set the Expropriation Correlation Number field of the affected virtual processor’s VPA to 0x0000 (see Table 194, “Firmware Written VPA Starting at Byte Offset 0x178,” on page 487).

R1–14.12.3.8.2.2–5. For the ESN option: When the partition accesses an expropriated page, the platform is not prevented by R1–14.12.3.8.2.1–1, does not associate a physical page with the logical page prior to returning control to the affected virtual processor, and the restoration of the logical page has NOT previously been reported to the OS with an expropriation notification, the platform must, set the Expropriation Correlation Number field of the VPA to a non-zero unique value for all outstanding recovering pages for the affected partition.

R1–14.12.3.8.2.2–6. For the ESN option: When the partition accesses an expropriated page, the platform is not prevented by R1–14.12.3.8.2.1–1, does not associate a physical page with the logical page prior to returning control to the affected virtual processor, and the restoration of the logical page has previously been reported to the OS with an expropriation notification, the platform must set the Expropriation Correlation Number field of the VPA to the same value as was supplied with the previous expropriation notification event associated with the outstanding recovering page for the affected partition.

R1–14.12.3.8.2.2–7. For the ESN option: When the partition performs an instruction fetch from an expropriated page, the platform must, unless prevented by R1–14.12.3.8.2.1–1, signal the affected virtual processor with an Expropriation Interrupt by returning to the affected virtual processor's interrupt vector at location 0x0400 with the processor's MSR, SRR0 and SRR1 registers set as if the instruction fetch had experienced a translation fault type of Instruction Storage Interrupt except that SRR1 bit 46 (Trap) is set to a one.

R1–14.12.3.8.2.2–8. For the ESN option: When the partition performs a load or store instruction that accesses an expropriated page, the platform must, unless prevented by R1–14.12.3.8.2.1–1, signal the affected virtual processor with an Expropriation Interrupt by returning to the affected virtual processor's interrupt vector at location 0x0300 with the processor's MSR, DSISR, DAR, SRR0 and SRR1 registers set as if the storage access had experienced a translation fault type of Data Storage Interrupt except that SRR1 bit 46 (Trap) is set to a one.

14.12.3.8.3 ESN Subvention Event Notification

ESN uses an event queue within the Subvention Notification Structure (SNS) to notify the OS of page subvention operations. Subvention events have a two byte SNS-EQ entry which has the value of the expropriation correlation number from the associated expropriation notification event

R1–14.12.3.8.3–1. For the ESN option: The platform must implement the structures, syntax and semantics described in Section 14.12.3.8.3.1, "SNS Memory Area," on page 488, Section 14.12.3.8.3.2, "SNS Registration (H_REG_SNS)," on page 489, and Section 14.12.3.8.3.3, "SNS Event Processing," on page 490.

14.12.3.8.3.1 SNS Memory Area

R1–14.12.3.8.3.1–1. For the ESN option: The platform must support the 4K byte aligned SNS not spanning its page boundary defined by Table 196, "Subvention Notification Structure," on page 488.

Table 196. Subvention Notification Structure

Access	Offset		Usage
Written by OS Read by Hypervisor	0x00	Bit	Notification Control
		0	Notification Trigger
		1-7	Reserved
Written by Hypervisor Read by OS	0x01	Bit	Event Queue State
		0	0 = Operational 1 = Overflow
		1-7	Reserved

Table 196. Subvention Notification Structure

Access	Offset	Usage
Set to non-zero by Hypervisor Read and cleared to zero by OS	0x02-0x02	First SNS EQ Entry
	. . .	
	(SNS Length -2) - SNS Length - 1)	Last SNS EQ Entry

14.12.3.8.3.2 SNS Registration (H_REG_SNS)

Syntax:

```
int64                   /* H_Success: Expected return code */
                        /* H_Function: The function is not allowed from the calling partition */
                        /* H_RESCINDED: The Address parameter refers to a rescinded shared logical resource */
                        /* H_Parameter: The Address parameter is invalid (4K aligned in the caller's memory)*/
                        /* H_P2: The Length parameter is odd or not within the limits of: */
                        /* (256 <= Length <= distance to Page Boundary) */
                        /* H_Hardware: The hcall() experienced a hardware fault potentially preventing the function */
    hcall   (const uint64 H_REG_SNS), /* Registers the SNS structure returning virtual interrupt parameters*/
            int64 Address,          /* Logical address of the SNS structure */
            uint64 Length);         /* Length of the SNS structure */
```

Semantics:

- If the Address parameter is -1 then deregister any previously registered SNS for the partition, disable ESN functions and return H_Success. (Care is required on the part of the OS not to create any Restoration Paradox Failures prior to registering a new SNS. See Section 14.12.3.8.4.2, “Restoration Paradox Failure,” on page 490 for details.)
- If the Shared Logical Resource option is implemented and the Address parameter represents a shared logical resource that has been rescinded, then return H_RESCINDED.
- If the Address parameter is not 4K aligned in the valid logical address space of the caller, then return H_Parameter.
- If the Length parameter is less than 256 or the Address plus Length spans the page boundary of the page containing the starting logical address, then return H_P2.
- Register the SNS structure for the calling partition by saving the partition specific information:
  - Record the SNS starting address
  - Record the SNS ending address
  - Record the next EQ entry to fill address (SNS starting address +2)
  - Set the SNS interrupt toggle = SNS Notification Trigger
  - Set the SNS Event Queue State to “Operational”
- Return:
  - R3: H_Success
  - R4: Value to be passed in the “unit address” parameter of the H_VIO_SIGNAL hcall() to enable/disable the virtual interrupt associated with the transition of the SNS from empty to non-empty.

- R5: Interrupt source number associated with the SNS empty to non-empty virtual interrupt.

14.12.3.8.3.3 SNS Event Processing

The following sequence is used by the platform to post an SNS event. The SNS-EQ used corresponds to the EEN event type. This sequence refers to fields described in Table 196, “Subvention Notification Structure,” on page 488.

1. If the SNS EQ overflow state is set, exit.
   /* An EQ overflow drops all new events until software recovers the EQ*/
2. Using atomic update protocol, store the event identifier into the location indicated by the SNS next EQ entry to fill pointer if the original contents of the location were zero; else set the associated EQ overflow state and exit.
   /* The value of zero is reserved for an unused entry -- an EQ overflow drops the new event */
3. Increment the SNS next EQ entry to fill pointer by the size of the EQ entry (2) modulo the size of the EQ
   /* Adjust fill pointer */
4. If the SNS interrupt toggle = SNS Notification Trigger then exit.
   /* Exit on no event queue transition */
5. Invert the SNS interrupt toggle.
   /* Remember event queue transition */
6. If the SNS interrupt is enabled, signal a virtual interrupt to the associated partition.
   /* Signal transition when enabled */

14.12.3.8.4 ESN Interrupts

The ESN option may generate several interrupts to the partition OS. Defined in this section are those in addition to the Expropriation Notification interrupts defined above.

14.12.3.8.4.1 Subvention Notification Queue Transition Interrupt

R1–14.12.3.8.4.1–1. For the ESN option: When the platform has restored the association of a physical page with the logical page that caused an Expropriation Notification interrupt with a non-zero Expropriation Correlation Number, the platform must post the corresponding Expropriation Correlation Number to the Subvention Event Queue see Section 14.12.3.8.3.3, “SNS Event Processing,” on page 490.

14.12.3.8.4.2 Restoration Paradox Failure

Restoration Paradox Failures result in an unrecoverable memory failure machine check.

R1–14.12.3.8.4.2–1. For the ESN option: When the platform finds that Expropriation Notification has been disabled after it has discarded the contents of an “Expendable” page, it must treat any access to such a page as an unrecoverable error restoring the association of a physical page with the expropriated logical page.

14.12.4 Virtual Partition Memory Pool Statistics Function Set

The hcall-vpm-pstat function set may be configured via the partition definition in none or any number of partitions as the VPM administrative policy dictates.

14.12.4.1 H_VPM_PSTAT

This hcall() returns statistics on the physical shared memory pool. Since these statistics can be manipulated by the processing of a single partition, there is a risk of creating a covert channel through this call. To mitigate this risk, the call is contained in a separate function set that can be protected by authorization methods outside the scope of LoPAPR.

Syntax:

```
int64                   /* H_Success: Expected return code */
                        /* H_Function: The function is not allowed from the calling partition */
                        /* H_Authority: The calling partition is not authorized to make the call at this time */
                        /* H_Hardware: The hcall() experienced a hardware fault potentially preventing */
                                /* the function */
        hcall (const uint64 H_VPM_PSTAT); /* Returns the memory pool performance statistics */
```

Parameters:

- Input: None
- Output:
  - R4: Total VM Pool Page Faults
  - R5: Total Page Fault Wait Time (Time Base Ticks)
  - R7: Total Pool Physical Memory
  - R8: Total Pool Physical Memory that is I/O mapped
  - R9: Total Logical Real Memory that is Virtualized by the VM Pool

Semantics:

- Verify that calling partition has the authority to make the call else return H_Authority.
- Report the statistics for the memory pool used to instantiate the virtual real memory of the calling partition.
  - Place in R4 the summation of the virtual partition memory page faults against the memory pool since the initialization of the pool.
  - Place in R5 the summation of timebase ticks spent waiting for the page faults indicated in R4.
  - Place in R6 the total amount of physical memory in the memory pool.
  - Place in R7 the summation of the entitlement of all active partitions served by the memory pool.
  - Place in R8 the summation of the I/O mapped memory of all active partition served by the memory pool.
  - Place in R9 the summation of the logical real memory of all active partitions served by the memory pool.
- Return H_Success.

14.13 Logical Partition Control Modes

Selected logical partition control modes may be modified by the client program.

14.13.1 Secondary Page Table Entry Group (PTEG) Search

The page table search algorithm, described by the *Power ISA* [1], consists of searching for a Page Table Entry (PTE) in up to two PTEGs. The first PTEG searched is the “primary PTEG”. If a PTE match does not occur in the primary PTEG, the hardware may search the “secondary PTEG”. If a PTE match is not found in the searched PTEGs, the hardware signals a translation exception.

Code is not required to place any PTEs in secondary PTEGs. Therefore, if a PTE match does not occur in a primary PTEG there is no need for the hardware to search a secondary PTEG to determine that a search has failed. The "Secondary Page Table Entry Group" bit of **"ibm,client-architecture-support"** allows code to indicate that there is no need to search secondary PTEGs to determine that a PTE search has failed.

14.14 Partition Energy Management Option (PEM)

This section describes the functional interfaces that are available to assist the partition Operating System optimize trade offs between energy consumption and performance requirements.

14.14.1 Long Term Processor Cede

To enable the hypervisor to effectively reduce the power draw from unused partition processors, the concept of cede wakeup latency is introduced with the Partition Energy Management Option. A one byte cede latency specifier VPA field communicates the maximum latency class that the OS can tolerate on wakeup from H_CEDE. In general the longer the wakeup latency the greater the savings that can be made in power drawn by the processor during a cede operation. However, due to implementation restrictions, the platform might be unable to take full advantage of the latency that the OS can tolerate thus the cede latency specifier is considered a hint to the platform rather than a command. The platform may not exceed the latency state specified by the OS. Calling H_CEDE See Section 14.11.3.3, "H_CEDE," on page 460, with value of the cede latency specifier set to zero denotes classic H_CEDE behavior. Calling H_CEDE with the value of the cede latency specifier set greater than zero allows the processor timer facility to be disabled (so as not to cause gratuitous wake-ups – the use of H_PROD, or other external interrupt is required to wake the processor in this case). An External interrupt might not awake the ceded process at some of the higher (above the value 1) cede latency specifier settings. Platforms that implement cede latency specifier settings greater than the value of 1 implement the cede latency settings system parameter see Section 7.3.16.18, "Cede Latency Settings Information," on page 230. The hypervisor is then free to take energy management actions with this hint in mind.

R1–14.14.1–1. For the PEM option: The platform must honor the OS set cede latency specifier value per the definition of Section 14.14.1, "Long Term Processor Cede," on page 492.

R1–14.14.1–2. For the PEM option: The platform must map any OS set cede latency specifier value into one of its implemented values that does not exceed the latency class set by the OS.

R1–14.14.1–3. For the PEM option: The platform must implement the cede latency specifier values of 0 and 1 per Section 14.14.1, "Long Term Processor Cede," on page 492.

R1–14.14.1–4. For the PEM option: If the platform implements cede latency specifier values greater than 1 it must implement the cede latency settings values sequentially without holes.

R1–14.14.1–5. For the PEM option: If the platform implements cede latency specifier values greater than 1 each sequential cede latency settings value must represent a cede wake up latency not less than its predecessor, and no less restrictive than its predecessor.

R1–14.14.1–6. For the PEM option: If the platform implements cede latency specifier values greater than 1 it must implement the cede latency settings system parameter see Section 7.3.16.18, "Cede Latency Settings Information," on page 230.

14.14.2 H_GET_EM_PARMS

This hcall() returns the partition's energy management parameters. The return parameters are packed into registers.

Programming Note: On platforms that implement the partition migration option, after partition migration:

1. The support for this hcall() might change, the caller should be prepared to receive an H_Function return code indicating the platform does not implement this hcall().
2. Fields that were defined as “reserved” might contain data; calling code should be tested to ensure that it ignores fields defined as “reserved” at the time of its design, and that it operates properly when encountering “zeroed” defined fields that indicate that the field does not contain useful data.

Implementation Note: To aid the testing of calling code, implementations would do well to include debug tools that seed reserved return fields with random data.

Syntax:

```
int64                   /* H_Success: Expected return code */
                        /* H_Hardware: The hcall() experienced a hardware fault potentially preventing the function */
    hcall (const uint64 H_GET_EM_PARMS); /*Returns in R4 – R9 the Platform Energy Management Parameters. */
```

Parameters: (on return)

Register R3:

Return code

Register R4:

Platform	Group	Pool	Partition

- Status Codes (bit offset within 2 byte field): Bits 0:5 Reserved (zero)
- Bits 6:8 Energy Management major code:
  - 0b000: Non – floor modes:
    - Bits 9:15 Energy Management minor code:
      - 0x00: The energy management policy for this aggregation level is not specified.
      - 0x01: Maximum Performance (Energy Management enabled – performance may exceed nominal)
      - 0x02: Nominal Performance (Energy Management Disabled)
      - 0x03: Static Power Saving Mode
      - 0x04: Deterministic Performance (Energy Management enabled – consistent performance on a given workload independent of environmental factors and component variances)
      - 0x05 – 0x7F Reserved
  - 0b001: Dynamic Power Management:
    - Bits 9:15 Performance floor as a percentage of nominal (0% - 100%).
  - 0b010:0b111 Reserved

Implementation Note: Status Code Fields are determined by means outside the scope of LoPAPR. Platform designs may define a hierarchy of aggregations in which lower levels by default inherit the energy management policy of their parent.

Register R5:

Platform Power Draw	Reserved 0x00000000

A value of all zeros indicates that no Power Draw limit is set

- Bytes 0:3 four byte Power Draw Status/Limit for the platform
  - Bit 0: Power Draw Limit is hard/soft: 0 = Soft, 1 = Hard
  - Bits 1:7 Reserved.
  - Bits 8:31 unsigned binary Power Draw Limit times 0.1 watts

Register R6:

Reserved

Register R7:

Total processor energy consumed

- The total processor energy consumed by the calling partition since boot in Joules times 2**-16. The value zero indicates that the platform does not support reporting this parameter.

Register R8:

Total memory energy consumed

- The total memory energy consumed by the calling partition since boot in Joules times 2**-16. The value zero indicates that the platform does not support reporting this parameter.

Register R9:

Total I/O energy consumed

- The total I/O energy consumed by the calling partition since boot in Joules times 2**-16. The value zero indicates that the platform does not support reporting this parameter.

Semantics:

- Place the partition’s performance parameters for the calling virtual processor’s partition into the respective registers:
  - R4: Energy Management Status Codes
  - R5: Power Draw Limits (Platform and Group)
  - R6: Power Draw Limits (Pool and Partition)
  - R7: Partition Processor Energy Consumption

- R8: Partition Memory Energy Consumption
- R9: Partition I/O Energy Consumption

- Return H_Success.

R1–14.14.2–1. For the PEM option: The platform must implement the H_GET_EMP hcall() following the syntax and semantics of Section 14.14.2, “H_GET_EM_PARMS,” on page 492.

14.14.2.1 H_BEST_ENERGY

This hcall() returns a hint to the caller as to the probable impact toward the goal of minimal platform energy consumption for a given level of computing capacity that would result from releasing or activating various computing resources. The returned value is a unitless priority, the lower the returned value; the more likely the goal will be achieved. The accuracy of the returned hint is implementation dependent, and is subject to change based upon actions of other partitions; thus the implementation can only provide a “best effort” to be “substantially correct”. Implementation dependent support for this hcall() and supported resource codes might change during partition suspension as in partition hibernation or migration; the client program should be coded to gracefully handle H_Function, H_UNSUPPORTED, and H_UNSUPPORTED_FLAG return codes.

H_BEST_ENERGY may be used in one of two modes, “inquiry” or “ordered” specified by the setting of bit 54 of the eflags parameter. It is intended that ordered mode be used when the client program is largely indifferent to the specific resource instance to be released or activated. In ordered mode, H_BEST_ENERGY returns a list of resource instances in the order from the best toward worst to choose to release/activate to achieve minimal energy consumption starting with an initial resource instance in the ordered list (if the specified initial resource is the reserved value zero the returned list starts with the resource having the greatest probability of minimizing energy consumption). It is intended that inquiry mode be used when the client program wishes to compare the energy advantage of making a resource selection from among a set of candidate resource instances. In inquiry mode, H_BEST_ENERGY returns the unitless priority of releasing/acquiring each of the specified resource instances. It is expected that in the vast majority of cases, the client code will receive data on a sufficient number of resource instances in one H_BEST_ENERGY call to make its activate/release decision; however, in those rare cases where more information is needed, a series of H_BEST_ENERGY calls can be made to accumulate information on an arbitrary number of computing resource instances.

Platforms may optionally support “buffered ordered” return data mode. If the platform supports “buffered ordered” return data mode, a “b” suffix appears at the end of the list that terminates the hcall-best-energy-1 function set entry. If the “buffered ordered” return data mode is supported the caller may specify the “B” bit in the eflags parameter and supply in P3 the logical address of a 4K byte aligned return buffer.

The probable effects of a given resource instance selection might vary depending upon the intention of the client program to take other actions. These other actions include the ability to reactivate a released resource within a given time latency and number of resources the client program intends to activate/release as a group. The eflags parameter to H_BEST_ENERGY contains fields that convey hints to the platform of the client program intentions in these areas; implementations might take these hints into consideration as appropriate. The high order four (4) bytes of the eflags parameter contain the unsigned required reactivation latency in time base ticks (the reserved value of all zeros indicates an unspecified reactivation latency).

Calling H_BEST_ENERGY with the eflags “refresh” flag (bit 54) equal to a one causes the hypervisor to compute the relative unitless priority value (1 being the best to activate/release with increasing numbers being poorer choices from the perspective of potential energy savings) for each instance of the specified resource that is owned by the calling partition. If the hypervisor can not distinguish a substantially different estimate for the various resource instances the call returns H_Not_Available. If the “refresh” flag is equal to a zero, the list as previously computed is used. Care should be exercised when using the non-refresh version to ensure that the state of the partition’s owned resource list has been initialized at some point and has not changed due to resource instance activation/release (including dynamic reconfigura-

tion) activities by other partition threads else the results of the H_BEST_ENERGY call are unpredictable (ranging from inaccurate prediction values up to and including error code responses).

The return values for H_BEST_ENERGY are passed in registers. Following standard convention, the return code is in R3. Register R4 contains the response count. If the call is made in "inquiry" mode the response count equals the number of non-zero requested resource instance entries in the call. If the call is made in "ordered" mode, the response count contains the number of entries in the ordered list from the first entry returned until the worst choice entry. If the response count is <= 8 (512 for ordered buffer mode) then the response count also indicates how many resource instances are being reported by this call, if the response count is >8 (512 for ordered buffer mode) then this call reports eight (512 for ordered buffer mode) resource instances. Each response consists of three fields: bytes 0 -- 2 are reserved, byte 3 contains the unitless priority for selecting the indicated resource instance, and bytes 4 -- 7 contain the resource instance identifier value corresponding to that passed in the **"ibm,my-drc-index"** property.

In order to represent more accurately the significance of certain priority values relative to others, the platform might leave holes in the ranges of reported priority values. As an example there may be a gap of several priority numbers between the value associated with a resource that can be powered down versus one that can only be placed in an intermediate energy mode, and yet again another gap to a resource that represents a necessary but not sufficient condition for reducing energy consumption.

Syntax:

```
Int64                   /* H_Success: Expected return code */
                        /* H_Hardware: The hcall() experienced a hardware fault potentially preventing the function*/
                        /* H_Function: The hcall() is not supported */
                        /* H_Busy: The hcall() is not complete call again */
                        /* H_Not_Available: Differentiated energy estimates are not available for this resource */
                        /* H_UNSUPPORTED_FLAG: Unsupported eflags parameter bits (32 -- 39 & 48 -- 56) */
                        /* H_UNSUPPORTED: The specified resource code is not supported by */
                        /* this implementation*/
                        /* H_P2 -- H_P9 Invalid resource identifier value for the calling partition */
    hcall       (uconst64 H_BEST_ENERGY,
                        int64 eflags,       /* Bits 0 -- 31 Required wakeup latency in time base ticks. */
                                            /* Bits 32 -- 39 Reserved for expansion */
                                            /* Bits 40 -- 47 1 Byte count of the number of resources that the */
                                            /* caller intends to activate / release*/
                                            /* Bits 48 -- 51 Reserved for expansion */
                                            /* Bit 52 = 0b0 return in registers = 0b1 ordered buffer mode
                                            /* Bit 53 = 0b0 use established list = 0b1 refresh list */
                                            /* Bit 54 = 0b0 inquiry; = 0b1 ordered */
                                            /* Bit 55 = 0b0 release; = 0b1 activate resource */
                                            /* Bits 56 -- 63 resource code: */
                                                    /* 0 Reserved */
                                                    /* 1 Processor */
                                                    /* 2 Memory LMB */
                        int64 P2,           /* The parameters P2 -- P9 are all the same format, */
                                            /* except in ordered buffer mode when P3 contains the logical */
                                            /* address of a 4K byte aligned caller partition memory buffer. */
                        int64 P3,           /* On input they contain either the reserved value of zero or */
                        int64 P4,           /* the resource instance identifier value as reported in the */
                        int64 P5,           /* "ibm,my-drc-index" property. */
                        int64 P6,           /* In "inquiry" mode they list resource instances */
                        int64 P7,           /* queried; from the contents of P2 up to the first parameter */
                        int64 P8,           /* containing all zeros – from there on to P9 all the rest are */
                        int64 P9);          /* ignored. */
```

```
/* In “ordered” mode P2 contains the resource instance */
/* identifier of the first resource instance to be reported */
/* the reserved value of zero indicates the list starts from the */
/* best resource instance available – from there on to P9 all */
/* the rest are ignored. */
```

Parameters: (on entry)

eflags

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Required wake up latency				Reserved	Number of Resources	rrrrBROA	Resource Code

r=reserved B=Buffered R=Refresh O=Ordered A=Active

P2 -- P9

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Reserved				Resource ID (as in “ibm,my-drc-index”)			

R5 -- R12

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Reserved			Priority	Resource ID (as in “ibm,my-drc-index”)			

(on return)

- R3: Return code
- R4: Response Count Value <8 indicate the number of returned values in registers starting with R5. The contents of registers after the last returned value as indicated by the Response Count Value are undefined.
- R5 -- R12 Bytes 0-2 Reserved
- Byte 3: 1 – 255 -- unitless priority value relative to lowest total energy consumption for selecting the corresponding resource ID.
- Bytes 4-7 Resource instance ID to be used as input to dynamic reconfiguration RTAS calls as would the value presented in the **“ibm,my-drc-index”** property.

Semantics:

- If the resource code in the eflags parameter is not supported return H_UNSUPPORTED
- If other binary eflags values are not valid then return H_UNSUPPORTED_FLAG with the specific value being (-256 – the bit position of the highest order unsupported bit that is a one);
- If the eflags parameter “refresh” bit is zero and the list has not been refreshed since the last return of H_Not_Available then return H_Not_Available.
- If the eflags parameter “refresh” bit is a one then:

- If energy estimates for the partition owned resources are substantially indistinguishable then return H_Not_Available.
- Assign a priority value to each resource of the type specified in the resource code owned by the calling partition relative to the probable effect that selecting the specified resource to activate/release (per eflags code) within the specified latency requirements would have on achieving minimal platform energy consumption. (1 being the best increasing values being worse – implementations may choose to use an implementation dependent subset of the available values)
- Order the specified resources owned by the calling partition starting with those having a priority value of 1; setting the resource pointer to reference that starting resource.

◆ If the eflags parameter bit 54 is a one (“ordered”) then

- If P2 == 0 then set pointer to best resource in ordered list
- Else
  - ❑ If P2 <> the drc-index of one of the resources in the ordered list then return H_P2
  - ❑ Else set pointer to the resource corresponding to P2
- Set R4 to the number of resources in the ordered list from the pointer to the end
- If eflags “B” bit == 0b0 then /* this assumes that the ordered buffer option is supported */:
  - ❑ If R4 > 8 set count to 8 else set count to R4
  - ❑ Load “count” registers starting with R5 with the priority value and resource IDs of the “count” resource instances from the ordered list starting with the resource instance referenced by “pointer”.
- Else
  - ❑ If P3 does not contain the 4K aligned logical address of a calling partition memory page then return H_P3
  - ❑ If R4 > 512 set count to 512 else set count to R4
  - ❑ Load “count” 8 byte memory fields starting with logical address in R3 with the priority value and resource IDs of the “count” resource instances from the ordered list starting with the resource instance referenced by “pointer”.
- Return H_SUCCESS

◆ Else /* “inquiry” mode */

- Set R4 to zero
- For each input parameter P2 -- P9 or until the input parameter is zero
  - ❑ If the input parameter Px <> the drc-index of one of the resources in the ordered list then return H_Px
  - ❑ Fill in byte 3 of the register containing Px with the priority value of the resource instance corresponding to the drc-index (bytes 4 -- 7) of the register.
  - ❑ Increment R4
- Return H_SUCCESS

R1–14.14.2.1–1. For the PEM option: The platform must implement the H_BEST_ENERGY hcall() following the syntax and semantics of Section 14.14.2.1, “H_BEST_ENERGY,” on page 495.

14.15 Platform Facilities

This section documents the hypervisor interfaces to optional platform facilities such as special purpose coprocessors.

14.15.1 H_RANDOM

If the platform supports a random number generator platform facility the **"ibm,hypertasfunctions"** property of the **/rtas** node contains the function set specification "hcall-random" and the following hcall() is supported.

```
int64                   /* H_Success Expected Return code */
                        /* H_Hardware The hcall() experienced a hardware fault potentially preventing the function */
        hcall (const H_RANDOM) /* Returns a random number in R4 */
```

14.15.2 Co-Processor Facilities

If the platform supports a co-processor platform facility the **"ibm,hypertas-functions"** property of the **/rtas** node contains the function set specification "hcall-cop" and the following hcall()s are supported.

For asynchronous coprocessor operations the caller may either specify an interrupt source number to signal at completion or the caller may poll the completion code in the CSB. The hypervisor and caller need to take into account the processor storage models with explicit memory synchronization to ensure that the rest of the return data from the operation is visible prior to setting the CSB completion code, and that any operation data that might have been fetched prior to the setting of the CSB completion code is discarded.

Note: The H_MIGRATE_DMA hcall() does not handle data pages subject to co-processor access, it is the caller's responsibility to make sure that outstanding co-processor operations do not target pages that are being migrated by H_MIGRATE_DMA.

14.15.2.1 14.15.2.1 H_COP_OP:

The architectural intent of this hcall() is to initiate a co-processor operation. Co-processor operations may complete with either synchronous or asynchronous notification. In synchronous notification, all platform resources associated with the operation are allocated and released between the call to H_COP_OP and the subsequent return. In asynchronous notification, operation associated platform resources may remain allocated after the return from H_COP_OP, but are subsequently recovered prior to setting the completion code in the CSB. For the partition migration option no asynchronous notification operation may be outstanding at the time the partition is suspended.

```
int64                   /* H_Success Expected Return code */
                        /* H_RH_PARM Invalid resource id handle for the caller */
                        /* H_UNSUPPORTED_FLAG Reserved flags field bit is non-zero */
                        /* H_ST_PARM Invalid operation specification.*/
                        /* H_OP_MODE Function code invalid in synchronous notification */
                        /* H_TOO_BIG The specified Input stream is too long */
                        /* or can not be completed synchronously*/
                        /* H_OVERLAP There exists an unsupported overlap among passed buffer areas */
                        /* H_NOT_ENOUGH_RESOURCES For the CMO option and asynchronous */
                        /* operations the memory entitlement is exhausted */
                        /* H_RESCINDED a data/status area references a rescinded shared logical resource */
                        /* H_P2 Invalid in parameter */
                        /* H_P3 Invalid inlen parameter */
                        /* H_P4 Invalid out parameter */
                        /* H_P5 Invalid outlen parameter */
```

```
                /* H_P6 Invalid csbcpb parameter */
                /* H_SG_LIST Invalid Scatter/Gather List element */
                /* H_Resource Insufficient hypervisor resources to perform function */
                /* H_Busy The hardware is busy user may call back later */
                /* H_Hardware The hcall() experienced a hardware fault potentially preventing the function */
hcall (const H_COP_OP,
        uint64 flags,       /* sub functions and modifiers: */
                /*              Bit Number(s) */
                /*              0 - 38 = 0b0s Reserved for function expansion */
                /*              39 =1 on Asymmetric Encryption operations indicating that the */
                /*                      High order 16 bits of the “in” parameter contain the */
                /*                      “Rc” field specifying the encoded operand length */
                /*                      while the remainder of the “in” and “inlen” */
                /*                      parameter bits are reserved and should be 0b0 */
                /*              40-41 Notification of operation */
                /*                      00 Synchronous: Hypervisor waits for completion */
                /*                      01 Reserved */
                /*                      10 Asynchronous: Hypervisor returns after start */
                /*                      11 Async Notify: Hypervisor starts with interrupt */
                /*              42-55 For Async Notify = index of the interrupt */
                /*                      descriptor to be used to signal completion */
                /*                      else =0x0000 */
                /*              56-63 FC field */
        uint32 rid          /*Resource identifier as from the "ibm,resource-id" property*/
        int64 in,           /*Input data block logical real address */
        int64 inlen         /*If non negative the length of the input data block, */
                            /*If negative the length of the input data descriptor list in bytes */
        int64 out           /*Output data block logical real address */
        int64 outlen,       /*If non negative the length of the output data block, */
                            /*If negative the length of the output data descriptor list in bytes */
        uint64 csbcpb       /*The logical real address of the 4k naturally aligned storage block */
                            /* containing the CSB & optional FC field specific CPB */
        );
```

Syntax:

- Flags:
  - Reserved (bits 0-- 38)
  - “Rc” (bit 39) On Asymmetric Encryption operations the “Rc” bit indicates that the high order 16 bits of the “in” parameter contain the “Rc” field specifying the encoded operand length while the remainder of the “in” and “inlen” parameter bits are reserved and should be 0b0
  - Notification of Operation (bits 40-- 41):
    - 00 Synchronous: In this mode the hypervisor synchronously waits for the coprocessor operation to complete. To preserve Interrupt service times of the caller and quality of service for other callers, the length of synchronous operations is restricted (see inlen parameter).
    - 01 Reserved
    - 10 Asynchronous: In this mode the hypervisor starts the coprocessor operation and returns to the caller. The caller may poll for operation completion in the CSB.

- 11 Async Notify: In this mode the hypervisor starts the coprocessor operation as with the Asynchronous notification above however the operation is flagged to generate a completion interrupt to the interrupt source number given in the **"ibm,copint"** property. When the interrupt is signaled the caller may check the operation completion status in the CSB.

- Interrupt descriptor index for Async Notify (bits 42-- 55)

- FC field: The FC field is the co-processor name specific function code (bits 56-- 63)

- Resource identifier (bits 32-- 63(as from the **"ibm,resource-id"** property))

- in/inlen and out/outlen parameters:

- If the *len parameter is non-negative; the respective in/out parameter is the logical real address of the start of the respective buffer. The starting address plus the associated length may not extend beyond the bounds of a 4K page owned by the calling partition. For synchronous notification operations, the parameter values may not exceed an implementation specified maximum; in some cases these are communicated by the values of the **"ibm,max-sync-cop"** property of the device tree node that represents the co-processor to the partition.

- If the *len parameter is negative; the respective in/out parameter is the logical real address of the start of a scatter/gather list that describes the buffer with a length equal to the absolute value of the *len parameter. The starting address of the scatter/gather list plus the associated length may not extend beyond the bounds of a 4K page owned by the calling partition. Further the scatter/gather list shall be a multiple of 16 bytes in length not to exceed the value of the "**ibm,max-sg-len"** property of the device tree node that represents the coprocessor to the partition. Each 16 byte entry in the scatter gather list consists of an 8 byte logical real address of the start of the respective buffer segment. The starting address plus the associated length may not extend beyond the bounds of a 4K page owned by the calling partition. For synchronous notification operations, the summation of the buffer segment lengths for the in scatter/gather list may be limited; in some cases these limitations are communicated by the value of the **"ibm,max-sync-cop"** property of the device tree node that represents the coprocessor to the partition.

- csbcpb: logical real address of the 4K naturally aligned memory block used to house the co-processor status block and FC field dependent co-processor parameter block.

- Output parameters on return

- R3 contains the standard hcall() return code: if the return code is H_Success then the contents of the 4K naturally aligned page specified by the csbspb parameter are filled from the hypervisor csb and cpb with addresses converted from real to calling partition logical real

Semantics:

- The hypervisor checks that the resource identifier parameter is valid for the calling partition else returns H_RH_PARM.

- The hypervisor checks that for the coprocessor type specified by the validated resource identifier parameter there are no non-zero reserved bits within the function expansion field of the flags parameter else returns H_UNSUPPORTED_FLAG for the highest order non-zero unsupported flag.

- If the operation notification is asynchronous, check that there are sufficient resources to initiate and track the operation else return H_Resource.

- The hypervisor checks that the flag parameter notification field is not a reserved value and FC field is valid for the specified coprocessor type else returns H_ST_PARM

- If the notification field is "synchronous" the hypervisor checks that the FC field is valid for synchronous operations else return H_OP_MODE.

- The hypervisor builds the CRB CCW field per the coprocessor type specified by the validated resource identifier parameter and by copying the coprocessor type defined number of FC field bits from the low order flags parameter FC field to the corresponding low order bits of CCW byte 3.
- If the resource ID is an asymmetric encryption then (If the Flags Parameter "Rc" bit is on then check the High order 16 bits of the "in" parameter for a valid "Rc" encoding and transfer to the CRB starting at byte 16 else return H_P2) else Validate the inlen/in parameters and build the source DDE
  - Verify that the "in" parameter represents a valid logical real address within the caller's partition else return H_P2
  - If the "inlen" parameter is non-negative:
    - Verify that the logical real address of (in + inlen) is a valid logical real address within the same 4K page as the "in" parameter else return H_P3.
    - If the operation notification is synchronous verify that the combination of parameter values request a sufficiently short operation for synchronous operation else return H_TOO_BIG.
  - If the "inlen" parameter is negative:
    - Verify that the absolute value of inlen meet all of the follow else return H_P3:
      - Is <= the value of **"ibm,max-sg-len"**
      - Is an even multiple of 16
      - That in + the absolute value of inlen represents a valid logical real address within the same 4K caller partition page as the in parameter.
    - Verify that each 16 byte scatter gather list entry meets all of the following else return H_SG_LIST:
      - Verify that the first 8 bytes represents a valid logical real address within the caller's partition.
      - Verify that the logical real address represented by the sum of the first 8 bytes and the second 8 bytes is a valid logical real address within the same 4K byte page as the first 8 bytes.
    - If the operation notification is synchronous verify that the sum of all the scatter gather length fields (second 8 bytes of each 16 byte entry) request a sufficiently short operation for synchronous operation else return H_TOO_BIG.
  - For the Shared Logical Resource Option if any of the memory represented by the in/inlen parameters have been rescinded then return H_RESCINDED.
  - Fill in the source DDE list from the converted the in/inlen parameters.
- Validate the outlen/out parameters and build the target DDE
  - Verify that the "out" parameter represents a valid logical real address within the caller's partition else return H_P4
  - If the "outlen" parameter is non-negative verify that the logical real address of (out + outlen) is a valid logical real address within the same 4K page as the "out" parameter else return H_P5.
  - If the "outlen" parameter is negative:
    - Verify that the absolute value of outlen meet all of the follow else return H_P5:
      - Is <= the value of **"ibm,max-sg-len"**
      - Is an even multiple of 16

- ❖ That out + the absolute value of outlen represents a valid logical real address within the same 4K caller partition page as the out parameter

- ❑ Verify that each 16 byte scatter gather list entry meets all of the following else return H_SG_LIST:

  - ❖ Verify that the first 8 bytes represents a valid logical real address within the caller's partition.

  - ❖ Verify that the logical real address represented by the sum of the first 8 bytes and the second 8 bytes is a valid logical real address within the same 4K page as the first 8 bytes.

- ■ For the Shared Logical Resource Option if any of the memory represented by the out/outlen parameters have been rescinded then return H_RESCINDED.

- ■ Fill in the destination DDE list from the converted the out/outlen parameters.

- ◆ If the operation notification is asynchronous then verify that the input and output buffers do not overlap else return H_OVERLAP (makes the operations transparently restartable)

- ◆ Check that the csbcpb parameter is page aligned within the calling address space of the calling partition else return H_P6

- ◆ If the operation specifies a CPB and the specified CPB is invalid for the operation then return H_ST_PARM.

- ◆ Set the CRB CSB address field & C bit to indicate a valid CCB

- ◆ If the operation notification is asynchronous notify, then:

  - ■ Check that the flags parameter interrupt index value is within the defined range for the validated rid and is not currently in use for another outstanding COP operation else return H_INTERRUPT.

  - ■ Set the CRB CM field to command a completion interrupt,.

  - ■ Set the job id field in the Co-processor Completion Block to command the signaling via the interrupt source number contained the interrupt specifier indicated by the interrupt index value.

  - ■ For the CMO option, if the number of entitlement granules pinned for this operation causes the partition memory entitlement to be exhausted then return H_NOT_ENOUGH_RESOURCES; else pin and record the entitlement granules used by this operation, and increment the partition consumed memory entitlement for the number of entitlement granules pinned for this operation.

- ◆ Set the completion code field in the passed (via csbcpb parameter) CSB to invalid (it is subsequently set to valid at the end of the operation just after the rest of the contents of the 4k naturally aligned page specified by the csbcpb parameter are filled).

- ◆ Issue icswx

- ◆ If busy response to icswx implementation dependent (may be null) retry after backoff based upon some usage equality/priority mechanisms else return H_Busy.

- ◆ If the operation notification is asynchronous then Return H_Success

- ◆ Wait for completion posting in CSB (CSB valid bit. 1)

- ◆ The contents of the 4K naturally aligned page specified by the csbcpb parameter are filled from the hypervisor csb and cpb with addresses converted from real to calling partition logical real

- ◆ Return H_Success.

14.15.2.2 14.15.2.2 H_STOP_COP_OP

The architectural intent of this hcall() is to terminate a previously initiated co-processor operation.

```
int64                   /* H_Success Expected Return code */
                        /* H_RH_PARM Invalid resource id handle for the caller */
                        /* H_Parameter Reserved flags field bit is non-zero */
                        /* H_RESCINDED a data/status area references a rescinded shared logical resource */
                        /* H_P3 Invalid csbcpb parameter */
                        /* H_Busy The hardware is busy user may call back later */
                        /* H_Hardware The hcall() experienced a hardware fault potentially preventing the
                                function */
    hcall   (const H_STOP_COP_OP,
             uint64 flags,      /* sub functions and modifiers: Bits 0-- 63 reserved */
             uint32 rid,        /* identifier as from the "ibm,resource-id" property*/
             uint64 csbcpb      /*The logical real address of the 4k page aligned storage block */
                                /* containing the CSB & optional FC field specific CPB */
             );
```

Semantics:

- Check the rid parameter for validity for the caller else return H_RH_PARM
- If any reserved flags parameter bits are non zero then return H_Parameter.
- Check the csbcpb parameter for pointing within the caller's partition and 4K aligned else return H_P3
- For the shared logical resource option if the csbcpb parameter references a rescinded shared logical resource then return H_RESCINDED
- If the csbcpb parameter is not associated with an outstanding coprocessor operation then return H_NOT_ACTIVE.
- Send a kill operation to the coprocessor handling the outstanding operation
- Wait for the outstanding kill operation to complete.
- For the CMO option, unpin any entitlement granules still pinned for this operation and decrement the consumed partition memory entitlement for the number of entitlement granules pinned for this operation.
- Return H_Success.

15 Non Uniform Memory Access (NUMA) Option

15.1 Summary of Extensions to Support NUMA

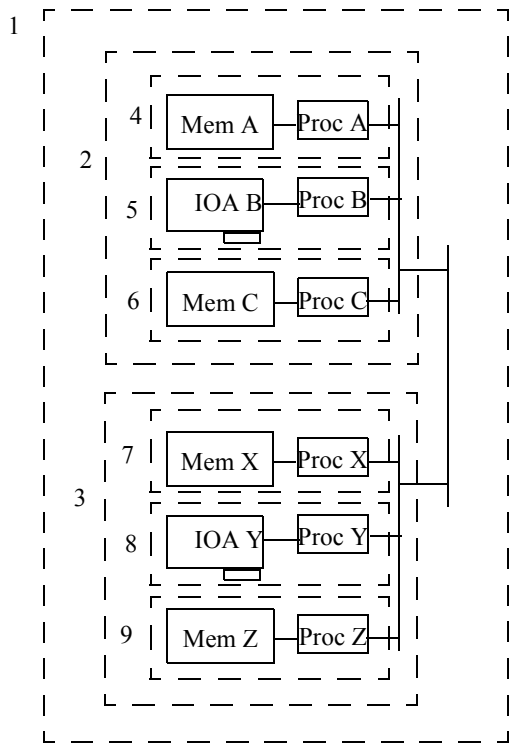
NUMA platforms to a first level approximation are simply a large scale Symmetric Multi-Processor. However to tune system performance and to aid in platform maintenance, the OS needs additional information and mechanisms. These include:

- Associativity -- to determine the platform resource groupings.
- Relative Performance Distances -- to determine the performance between resources within different groupings.
- Performance Monitor -- to provide usage data on the NUMA fabric.
- Dynamic Reconfiguration -- due to such causes as platform upgrade, reallocation of resources, or a repair of a failure.

There are two NUMA support options: the “NUMA” option and its proper subset the “Associativity Information” option.

15.2 NUMA Resource Associativity

Associativity Codes represent the groupings of the various platform resources into domains of substantially similar mean performance relative to resources outside of that domain. Resources subsets of a given domain that exhibit better performance relative to each other than relative to other resources subsets, are represented as being members of a sub-grouping domain. Such sub-domain grouping is represented to any level deemed significant by the platform design. Figure 14, “Example NUMA configuration with domains and corresponding “ibm,associativity” values,” on page 506 presents a simple system configuration with one possible decomposition into associativity domains. From the decomposition provided the **`“ibm,associativity”`** value string for each resource is enumerated.



Resource ibm,associativity

Mem A 1, 2, 4
Mem C 1, 2, 6
Mem X 1, 3, 7
Mem Z 1, 3, 9
Proc A 1, 2, 4
Proc B 1, 2, 5
Proc C 1, 2, 6
Proc X 1, 3, 7
Proc Y 1, 3, 8
Proc Z 1, 3, 9
IOA B 1, 2, 5
IOA Y 1, 3, 8

Figure 14. Example NUMA configuration with domains and corresponding **"ibm,associativity"** values

The OF Device Tree node for each allocable resource (processor, memory region, and IO slot) conveys information about the resources statically assigned to the client program; and contains the **"ibm,associativity"** property (see Section B.6.2.2, "Properties of the Children of Root," on page 679). This property allows the client program to determine the associativity between any two of it's resources. The greater the associativity the greater the expected performance when using those two resources in a given operation.

The legal form of the **"ibm,associativity"** property is dependent upon the setting of the **"ibm,architecture-vec-5"** property byte 5 bit 0. The bit value of zero allows the **"ibm,associativity"** property string to be sequenced in priority order; this form is being deprecated for new implementations in favor of the form indicated by the **"ibm,architecture-vec-5"** property byte 5 bit 0 having the value of one in which the **"ibm,associativity"** property string represents the strict physical hierarchy of the platform.

When the LPAR option is also implemented, the partition virtual resources may be mapped onto physical resources with in a very dynamic manor. Given that the resource mapping to the associativity domain is substantially consistent, the client program can make use of the associativity information to on the average optimize performance. If the resource mapping to the associativity domain is substantially inconsistent, then associativity information for the resources is not provided to prevent erroneous operation. If the long term mapping changes the client program can be made aware of the new associativity information using the *ibm,update-properties* RTAS call (See Section 7.4.8, "ibm,update-properties RTAS Call," on page 249).

R1–15.2–1. For the NUMA or Associativity Information option: The platform must include the **"ibm,associativity"** in the OF device tree **memory** node and the nodes of each processor, memory region, and PCI bridge onto which IOAs may be plugged if the component is dedicated to the partition. (The device tree node

for a component that the platform intends to virtualize should include an **"ibm,associativity"** property if the associativity domain information is substantially accurate.)

R1–15.2–2. For the NUMA option and SPLPAR option: In the case that both the NUMA and SPLPAR options are implemented, Requirement R1–15.2–1 is modified to remove processors from the list of system elements that must include the respective properties or interfaces described by that requirement. (The platform is encouraged to provide processor associativity information if it is substantially accurate.)

The **"ibm,associativity"** property contains one or more lists of numbers representing the resource's platform grouping domains. Each list, starts with a number representing the domain number of the highest level grouping within which the platform is capable of supporting direct access. This highest level may be a NUMA collective or possibly a cluster of machines with direct DMA access. Successive numbers represent sub-divisions of the previous higher level within which the expected mean value of the performance relative to outside resources is substantially similar. Implementations determine the number of levels that they report, subject to Requirements R1–15.2–1 and R1–15.2–3. The lowest level always being that of the allocable resource itself. The user of this information is cautioned not to imply any specific physical/logical significance of the various intermediate levels.

R1–15.2–3. For the NUMA or Associativity Information option: Differing levels of resource grouping represented in the **"ibm,associativity"** property must reflect statistically repeatable differences in the expected mean of measured performance.

R1–15.2–4. For the NUMA or Associativity Information option: The expected mean performance of any resource of a given type within the same grouping domain represented in the **"ibm,associativity"** property relative to resources outside of that grouping domain must be substantially similar.

The reason that the **"ibm,associativity"** property may contain multiple associativity lists is that a resource may be multiply connected into the platform. This resource then has a different associativity characteristics relative to its multiple connections. To determine the associativity between any two resources, the OS scans down the two resources associativity lists in all pair wise combinations counting how many domains are the same until the first domain where the two list do not agree. The highest such count is the associativity between the two resources.

15.3 Relative Performance Distance

An OS applies its NUMA tuning techniques based upon associativity and relative performance distance attributes. As a guide to relative performance distance, RISC Platforms provide the **"ibm,associativity-reference-points"** property. The information in this property represents a first order approximation to points having associativity and relative performance distance characteristics deemed to be of significant interest to optimizing client program performance.

The contents of the **"ibm,associativity-reference-points"** property is dependent upon the setting of the **"ibm,architecture-vec-5"** property byte 5 bit 0. The bit value of zero allows the **"ibm,associativity-reference-points"** property string to indicate logical structure points; this form is being deprecated for new implementations in favor of the form indicated by the **"ibm,architecture-vec-5"** property byte 5 bit 0 having the value of one in which the **"ibm,associativity-reference-points"** property string represents boundaries between associativity domains presented by the **"ibm,associativity"** property containing "near" and "far" resources.

R1–15.3–1. For the NUMA or Associativity Information option: The RTAS OF device tree node must contain the **"ibm,associativity-reference-points"**.

15.3.1 Form 0

When the **"ibm,architecture-vec-5"** property byte 5 bit 0 has the value of zero, the **"ibm,associativity-reference-points"** property defines reference points in the **"ibm,associativity"** property (see

Section B.6.3.1, “RTAS Node Properties,” on page 690) which roughly correspond to traditional notions of platform topology constructs. It is important for the user to realize that these reference points are not exact and their characteristics vary among implementations.

The first integer in the **“ibm,associativity-reference-points”** property relates the 1 based ordinal in the associativity lists of the platform’s **“ibm,associativity”** property associated with the traditional notion of a symmetric multi-processor within a NUMA platform. That is the level that represents building blocks of processors and memory that have the following characteristics:

- An OS is likely to view all members having roughly uniform access characteristics.
- Represents the highest level before an OS is likely to notice major Non-Uniformity of access.

The second integer in the **“ibm,associativity-reference-points”** property relates the 1 based ordinal in the associativity lists of the platform’s **“ibm,associativity”** property associated with the traditional notion of a processor group which is sometimes packaged in a multi-chip module. A processor group has similar characteristics to an SMP, however, several processor groups get packaged densely within the same physical enclosure forming an SMP. While the intra processor group accesses are measurably greater than inter processor group accesses they are a second order effect.

Subsequent *ibm,associativity-reference-points* entries are reserved.

15.3.2 Form 1

When the **“ibm,architecture-vec-5”** property byte 5 bit 0 has the value of one, the **“ibm,associativity-reference-points”** property indicates boundaries between associativity domains presented by the **“ibm,associativity”** property containing “near” and “far” resources. The first such boundary in the list represents the 1 based ordinal in the associativity lists of the most significant boundary, with subsequent entries indicating progressively less significant boundaries.

Note: Platforms are encouraged to report boundaries of actual significance. Thus if a platform has only a single significant boundary to report, the preferred form of the **“ibm,associativ¬ity-reference-points”** would contain a single entry. However, providing two or more entries that reference the same associativity domains provides equivalent information and is a legal representation.

15.4 Dynamic Reconfiguration with Cross CEC I/O Drawers

Should the configuration change in such a way that the associativity between an OS image’s resources changes, the platform notifies the OS via an event scan log. See Chapter 10, “Error and Event Notification,” on page 281.

R1–15.4–1. For the NUMA or Associativity Information option: If the platform configuration changes in such a way that the associativity between an OS image’s resources might have changed, the platform must notify the OS via an event scan or check exception log.

15.5 Maximum Associativity Domains

Since the number of associativity domains that a platform may exhibit is not apparent from the associativity properties presented at boot time, the platform provides the **“ibm,max-associativity-domains”** property in the **/rtas** node of the device tree (see Section B.6.3.1, “RTAS Node Properties,” on page 690).

R1–15.5–1. For the NUMA or Associativity Information option: The platform must provide the **“ibm,max-associativity-domains”** property in the **/rtas** node of the device tree.

15.6 Platform Resource Reassignment Notification Option (PRRN)

LoPAPR platforms that implement the LPAR option are allowed to transparently reassign the platform resources that are used by a partition. For instance, if a processor or memory unit is predicted to fail, the platform may transparently move the processing to an equivalent unused processor or the memory state to an equivalent unused memory unit. However, reassigning resources across NUMA boundaries may alter the performance of the partition. When such reassignment is necessary, the PRRN option provides mechanisms that inform the supporting OS of changes to the affinity among its platform resources. It is expected that handling such notifications will involve significant OS processing, therefore, changing affinity should be avoided, and when it is necessary to change the affinity of several of the resources owned by a partition, a single notification after all such changes have occurred is preferred.

The OS and platform firmware negotiate their mutual support of the PRRN option via the **`ibm,client-architecture-support`** interface (See Section B.6.2.3, “Root Node Methods,” on page 679). Should a partition be migrated from a platform that did not support the PRRN option, the target platform does not notify the partition’s OS of any PRRN events and, when possible avoids changing the affinity among the partition’s resources. Partitions that are about to be migrated complete/abort any in-process affinity change processing prior to the migration, and if the target platform does not support the PRRN option the partition will simply see no more PRRN events.

A PRRN event is signaled via the RTAS *event-scan* mechanism, which returns a Hot Plug Event message “fixed part” (See Section 10.3.2.1.9, “RTAS Event Return Format Fixed Part,” on page 292) indicating “Platform Resource Reassignment”. In response to the Hot Plug Event message, the OS may call *ibm,update-nodes* to determine which resources were reassigned, and then *ibm,update-properties* to obtain the new affinity information about those resources.

The PRRN *event-scan* RTAS message contains only the “fixed part” with the “Type” field set to the value 160 and no Extended Event Log. The four (4) byte Extended Event Log Length field is repurposed, since no Extended Event Log message is included, to pass the “scope” parameter that causes the *ibm,update-nodes* to return the nodes affected by the specific resource reassignment.

Requirements:

R1–15.6–1. For the PRRN Option: The platform must support the negotiation of the Associativity Information Option Control Platform Resource Reassignment Notification (Affinity Change) flag via the **`ibm,client-architecture-support`** interface.

R1–15.6–2. For the PRRN Option: If the client code did not claim support for the PRRN option via the **`ibm,client-architecture-support`** interface the platform must not present PRRN events per section Section 15.6, “Platform Resource Reassignment Notification Option (PRRN),” on page 509.

R1–15.6–3. For the combination of the PRRN and Partition Suspension Options: To avoid firmware function conflicts the client code must complete or abort any PRRN processing prior to exercising the Partition Suspension option.

R1–15.6–4. For the PRRN Option: The platform must inform the client code of platform resource reassignments via the *event-scan* RTAS mechanism with a “fixed part” only event return message as presented in Table 197, “RTAS Event Return Format (Fixed Part) for PRRN events,” on page 510

R1–15.6–5. For the PRRN Option: The platform must support the Platform Resource Reassignment scope (negative of the value contained in bits 32:64 of the RTAS Event Return Format (Fixed Part) for PRRN events) input parameter to input the *ibm,update-nodes* RTAS call.

Table 197. RTAS Event Return Format (Fixed Part) for PRRN events

Bit Field Name (bit number(s))	Description, Values (Described in Section 10.3.2.1, “Reporting and Recovery Philosophy, and Description of Fields,” on page 289})
Version (0:7)	A distinct value used to identify the architectural version of message
Severity (8:10)	EVENT (1)
RTAS Disposition (11:12)	FULLY_RECOVERED(0)
Optional_Part_Presence (13)	NOT_PRESENT (0): The optional Extended Error Log is not present.
Reserved (14:15)	0b00
Initiator (16:19)	HOT PLUG (6)
Target (20:23)	UNKNOWN (0): Not Applicable
Type (24:31)	Platform Resource Reassignment (160) – includes Change Scope in bits 32:63
Extended Event Log Length / Change Scope (32:63)	The scope parameter to be input the *ibm,update-nodes* RTAS to retrieve the nodes that were changed by selected “Hot Plug” events.

16 Service Indicators

This chapter defines service indicators[1] relative to:

- Which service indicators may be exposed to an OS and which may not
- The usage model for service indicators, regardless of whether they are exposed to the OS or not

16.1 General

This section gives some general background information required to understand the service indicator requirements. The service indicator requirements can be found starting in Section 16.2, “Service Indicator Requirements,” on page 524.

16.1.1 Basic Platform Definitions

The following are the definitions of some of the terms used in this architecture. See also the Glossary on page 891 , “Glossary,” on page 891.

16.1.1.1 “Enclosure”, Packaging, and Other Terminology

In order to abstract specific packaging differences between different products, this architecture uses a number of terms that denote a unit of packaging.

The term *enclosure* means something different, depending on the product line. Generally this is an entity that can be unplugged and be removed from the system, but may include the entire system, and generally encloses other FRUs. It is, however, possible to have a FRU that contains *one* other FRU, and not have it be an enclosure. See below, for more information.

The concept of the enclosure is very key to this architecture, because the enclosure provides the anchor point for the Enclosure Fault, Enclosure Identify, and (when applicable) the Error Log indicators.

- For a blade system, a base blade plus any attached sidecars.
  - A *sidecar* is a blade that plugs into a blade slot, but which is physically connected to the base blade and which cannot be removed without also removing the base blade and any other attached sidecars.
  - The Enclosure Identify indicator is located on the base blade. Sidecars do not have an Enclosure Identify indicator.
- A stand-alone computing box, like a deskside unit.
- A separately powered box that attaches to a stand-alone computing box (for example, an I/O expansion tower).

1.Note that many times “indicators” are referred to as “LEDs” as this is one of the most common implementations for indicators at the current time.

- For a rack system, a drawer or partial drawer, with its own power domains, within a rack system (but not a chassis, in blade system terms).

FRUs that have one or no internal FRUs are a possible exception to the above definition of enclosure. The general requirements are that the enclosing FRU does not need an Error Log indicator (see also Requirement R1–16.2.1.1–1), implements the full xipSIA Lightpath architecture *including FRU Identify*, has the enclosing FRU Fault/Identify and internal (if any) FRU Fault/Identify indicators visible from the outside of the system the same way that enclosure indicators would be, and rolls-up the FRU Fault/Identify indicators to the next level of indicators when there is a next level (for example, chassis level indicators). Examples of the type of FRUs that the xipSIA architecture team *might* approve as a non-enclosures is:

- Appliance drawers ("appliance" means that there are no field serviceable parts inside).
- Appliance Blades, except if they require an Error Log indicator.
- A power supply which comprises two or fewer FRUs.
- Fans, but not fan assemblies when the fan assemblies have three or more Fault indicators.

In addition, the term *System Enclosure* (also known as a *Primary Enclosure*) is used to denote the enclosure of a system that contains the one and only Error Log indicator[1] for the system. An enclosure that is not a System Enclosure is called a *secondary enclosure*. The System Enclosure is expected to be one that contains at least some of the system processors for the platform.

In this chapter, the term *chassis* will refer to a blade system chassis.

Other terminology used in this chapter includes:

activate — To activate an indicator (physical or virtual) means to set it to a non-off state (blink, blip, or on). An indicator does not need to be in the off (deactivated) state prior to being activated (for example a second request to activate an already active indicator, is also considered to be an activation of that indicator).

active state — An indicator in an active state is in a non-off state. Different indicator types can be set to a different set of active states. For each of the following indicators, the following states are applicable in the indicator active state (See Section 16.2.1.9, "Service Indicator State Diagrams," on page 533 for more detail on when each state is applicable and for the conditions under which a state transition is made):
FRU Identify: blink
FRU Fault: on
Blue Enclosure Identify: on or blink
Enclosure Fault: on or blip
Error Log: on
Blue Rack Identify: on
Blue Row Identify: on

blip — A blink state with a short duty cycle used in the "remind" state for Enclosure Fault Indicators. See also Requirement R1–16.2.1.7–4.

Chassis Enclosure Identify — An Enclosure Identify indicator at the blade system chassis level.

CRU — See FRU.

1.Previously known as the System Information (Attention) indicator.

deactivate	To deactivate an indicator (physical or virtual) means to set it to the off state. Deactivating a virtual indicator may or may not deactivate the physical indicator associated with that virtual indicator (see Section 16.2.1.9, “Service Indicator State Diagrams,” on page 533).
Enclosure Fault	An amber indicator which indicates, when activated, that there is a FRU Fault indicator in the enclosure that is active.
Enclosure Identify	An indicator that is used to identify an enclosure in an installation or an enclosure in a group. This indicator is blue in color and is turned on in the active identify state.
FRU	Field Replaceable Unit. Used to also mean CRU (Customer Replaceable Unit) in this chapter.
FRU Fault	An amber indicator that is used to point to a failing FRU in an enclosure.
FRU Identify	An amber indicator that is used to identify a FRU in an enclosure or a place where a FRU is to be plugged (for example, for an upgrade operation).
Guiding Light Mode	A platform implementation that provides FRU Identify indicators for identifying failing FRUs. See Section 16.1.1.4, “Service Indicator Modes,” on page 515 for more information.
ID	Shorthand used some places (mainly figures) in this chapter for “Identify” or “Identify indicator”.
Lightpath Mode	A platform implementation that provides FRU Fault indicators as the general way to identify failing FRUs. See Section 16.1.1.4, “Service Indicator Modes,” on page 515 for more information.
not visible to the OS	See transparent to the OS.
primary level indicators	The enclosure level indicators. For example, for rack systems, the enclosure is either the blade, for blade systems, or the drawer level, for non-blade systems.
roll-down	This term is not used by this architecture, but some people refer to roll-up as the action of activating a higher level indicator and roll-down as the action of deactivating a lower level indicator. This architecture will use roll-up for both activation and deactivation. See roll-up.
roll-up	The action of activating a higher level indicator, when a lower level indicator is activated, and deactivating it when all the lower level indicators that roll-up to that indicator are deactivated. For example, if a FRU Fault indicator is activated, it rolls-up and turns on the Enclosure Fault indicator, and when the last FRU Fault indicator in an enclosure is deactivated, the Enclosure Fault indicator for that enclosure is deactivated.
secondary level indicators	The indicators on levels below the Primary Level and above the FRU level.
SFP	Service Focal Point. See also Section 16.1.1.6, “Service Focal Point (SFP) and Service Partition,” on page 517.
Error Log	An amber indicator that indicates that the user needs to look at the error log or problem determination procedures, in order to determine the cause.
tertiary level indicators	The FRU level indicators.
transparent to the OS	Indicators whose state cannot be modified or sensed by OS or application level software. For example, power supply Fault indicators.
turn off	To turn off a physical indicator means exactly like it sounds. Turning off a virtual or logical indicator may or may not turn off the physical indicator, depending on the state diagram for the physical one (see Section 16.2.1.9, “Service Indicator State Diagrams,” on page 533).

visible to the OS — Indicators whose state can be modified or sensed by OS or application level software. For PCI Hot Plug indicators. See also Section 16.1.1.2, “Service Indicator Visibility and Transparency to the OS,” on page 514.

16.1.1.2 Service Indicator Visibility and Transparency to the OS

An indicator is said to be transparent or not visible to the OS when its state cannot be modified or sensed by OS or application level software (for example, power supply Fault indicators). An indicator is said to be visible to the OS when its state can be modified and sensed by OS or application level software (for example, PCI Hot Plug indicators).

Requirements on visibility can be found in Section 16.2, “Service Indicator Requirements,” on page 524.

16.1.1.3 Service Indicator

A *service indicator* is defined as any indicator that is used in the course of servicing a system. The intent of service indicators is *not*, in general, to increase system Error Detection and Fault Isolation (EDFI), but rather to guide the user in performance of a service action. Usages include (but are not limited to):

- Dynamic Reconfiguration (LoPAPR indicator type 9002) to indicate the status of DR operations on a Field Replacable Unit (FRU). More information on DR indicators can be found in Chapter 13, “Dynamic Reconfiguration (DR) Architecture,” on page 355. This indicator is amber[1] in color except for some legacy implementations which combined this indicator with the power indicator, where the color was green.
- An indication of a fault condition of a FRU (LoPAPR indicator type 9006, when OS visible). This indicator is amber in color. The FRU Fault indicator is handled differently by the platform based on whether or not the platform is Lightpath Mode platform or Guiding Light Mode:
  - For Guiding Light Mode platforms, FRU fault indicators are transparent to the software and therefore have some very specific requirements relative to their very localized behavior. In this case, although FRU Fault indicators themselves are transparent to the software, the associated failure itself, which would activate a FRU Fault indicator, will be available to the software that handles serviceable events.
  - For Lightpath Mode platforms, a FRU fault indicator will be available to the software and is activated by the detector of the error. In addition, the FRU Fault rolls up to an Enclosure Fault indicator.
- A system-wide indication of a fault or some condition needing attention in the system. An Error Log indicator (LoPAPR indicator type 9006) is an example of an OS-visible[2] indicator of this class of indicators. The Error Log indicator is a flag to the user that there is something in the system needing attention, and therefore a starting point to indicate that they should begin the isolation procedures to determine what needs attention. In a partitioned system, the physical Error Log[3] indicator may be the logical OR of individual virtual Error Log indicators (one virtual Error Log indicator per logical partition and one for each other separate entity that is non-partition related). This indicator is amber in color.
- An indication of an Identify (locate) operation. An Identify indicator (LoPAPR indicator type 9007) is an example of an indicator of this class. These indicators may or may not be visible to an OS. In this capacity, the indicator is activated[4] at the user’s request in order to help them locate a component in the system (for example, a FRU, a connector,

1.The term “amber” will be used in this chapter to mean any wavelength between yellow and amber.

2.For a definition of the visibility or transparency of an indicator, see Section 16.1.1.2, “Service Indicator Visibility and Transparency to the OS,” on page 514.

3.If the term “virtual” does not appear before “Error Log”, then the text is referring to the physical Error Log indicator.

4.For a definition of what “activate” means, see Section 16.1.1.1, ““Enclosure”, Packaging, and Other Terminology,” on page 511.

an enclosure, etc.). This indicator is amber in color, except for the Enclosure, Rack, and Row Identify indicators, which are blue in color.

- An indication of the power state of an entity. This indicator is platform controlled and is transparent to the OS(s). In addition to the power state, this indicator may be used to indicate a power failure or fault. This indicator is green in color.
- Environmental indicators such as ambient temperature too high. These are transparent to the OS.
- Hardware only indicators such as Ethernet activity indicators. These are transparent to the OS.

16.1.1.4 Service Indicator Modes

There are two modes that a platform can operate in relative to service indicators: Lightpath Mode and Guiding Light Mode. Any particular *platform* operates in one and only one mode relative to service indicators: Lightpath Mode or Guiding Light Mode. A *component* (hardware, firmware, or software) which is designed to be used in both a Lightpath Mode and a Guiding Light Mode platform needs to be able to operate in both modes.

For guidance in which mode a platform should be designed to operate, see Section 16.1.2, “Machine Classes and Service Strategy,” on page 518.

The following sections give an overview of these two modes. For specific requirements of each mode, see Section 16.2, “Service Indicator Requirements,” on page 524.

16.1.1.4.1 Lightpath Mode

The *Lightpath Mode* specifies a platform implementation of service indicators much like what industry-standard servers originally implemented with its Lightpath, except that FRU indicators also implement an Identify state along with the current Fault state. The Identify state overrides the Fault state while the Identify is active for an indicator, and the indicator is put into the Fault state, if one is pending, when the Identify is removed.

A summary of the Lightpath Mode is as follows (see Section 16.2, “Service Indicator Requirements,” on page 524 for detailed requirements):

- FRU Fault indicators are used as the general way to identify failing FRUs.
- The physical indicator that implements the Fault indicator states also implements the Identify indicator states (that is, a FRU Fault indicator is also a FRU Identify indicator for the same FRU).
- This mode is basically a superset of the Guiding Light Mode.
- FRU Fault indicators are presented to the OS for FRUs for which the OS image is expected to detect errors for either the entire FRU or part of the FRU. In the latter case, this represents a shared FRU, in which case the FRU Fault indicator is virtualized, so that one partition cannot view the setting by another partition, which would allow a covert storage channel (see also Section 16.1.1.5, “Covert Storage Channels,” on page 516).
- The OS and firmware are responsible for activating the FRU Fault indicator for a FRU for which they detect an error. Fault indicators are reset by the service action on the failing part that they represent.
- FRU Identify indicators are presented to the OS for FRUs that are fully owned by the OS image. They may also be presented for FRUs that are partially owned by the OS image. Ownership of a FRU by the OS image is defined as being the condition of the FRU being under software control by the OS, a device driver associated with the OS, or application software running on the OS. In the partially owned case, this represents a shared FRU, in which case the FRU Identify indicator is virtualized, so that one partition cannot view the setting by another partition, which would allow a covert storage channel (see also Section 16.1.1.5, “Covert Storage Channels,” on page 516).

- Connector Identify indicators are presented to the OS for connectors that are fully owned by the OS image. Ownership of a connector by the OS image is defined as being the condition of the connector being under software control by the OS, a device driver associated with the OS, or application software running on the OS.
- Enclosure Identify indicators are available to the OS when the OS fully owns a FRU in the enclosure. This indicator is virtualized in a partitioned system, so that one partition cannot view the setting by another partition, which would allow a covert storage channel.
- The Error Log indicator or virtual copy thereof (for LPARed platforms) is available to each OS image.

See Requirement R1–16.2.1.1–1 for more information for requirements on activation of the Fault indicators.

The Triple-S UI, when implemented, also adds additional requires to Lightpath Mode implementations. See Section 16.2.3, “Lightpath User Interface (UI) Requirements,” on page 544.

16.1.1.4.2 Guiding Light Mode

A summary of the Guiding Light Mode is as follows (see Section 16.2, “Service Indicator Requirements,” on page 524 for detailed requirements):

- FRU Identify indicators are used as the way of identifying service procedures like repair, reconfiguration, and upgrade. FRU Identify indicators are activated/deactivated by user via a user interface, to identify the FRU(s) involved in the service procedure.
- FRU Identify indicators are presented to the OS for FRUs that are fully owned by the OS image. Ownership of a FRU by the OS image is defined as being the condition of the FRU being under software control by the OS, a device driver associated with the OS, or application software running on the OS.
- Connector Identify indicators are presented to the OS for connectors that are fully owned by the OS image. Ownership of a connector by the OS image is defined as being the condition of the connector being under software control by the OS, a device driver associated with the OS, or application software running on the OS.
- Fault indicators are allowed, but not required, but if provided, must be transparent to any OS image, and are reset by the service action on the failing part that they represent. To be transparent to an OS means that they cannot be controlled by the OS, nor will they interfere with any other indicator that is controlled by the OS.
- Enclosure Identify indicators are provided as part of the Identify roll-up. Enclosure Identify indicators are available to the OS when the OS fully owns a FRU in the enclosure. This indicator is virtualized in a partitioned system, so that one partition cannot view the setting by another partition, which would allow a covert storage channel.
- The Error Log indicator, or a virtual copy thereof (for LPARed platforms), is available to the OS.

16.1.1.5 Covert Storage Channels

A *covert storage channel* is a path between two entities that can be used to pass data outside the normal data sharing paths like LANs. For example, if two OS images were given access to the same physical indicator and if each OS image could read the state of the indicator, then the indicator can become a single-bit covert storage channel between cooperating entities in the two OS images, to pass data back and forth. This cannot be allowed, for security reasons, and therefore this architecture defines the concept of virtual indicators.

A *virtual* indicator is provided to each OS image for each physical indicator that is shared between OS images. The physical indicator is activated when any virtual indicator for that physical indicator is activated and the physical indicator is deactivated when all virtual indicators for that physical indicator are deactivated. The general OS image can sense what it is trying to set the indicator to, but cannot sense what the other virtual indicators are set to, and hence no covert storage channel exists. The exception to the shared access is by a trusted Service Focal Point (see Section 16.1.1.6, “Service Focal Point (SFP) and Service Partition,” on page 517 for more details). For more informa-

tion on how virtual indicators affect the physical indicator state, see the physical indicator state diagrams later in this chapter.

An OS image in a partitioned system needs to realize that it may not have full control over all physical indicators to which it has access (that is, needs to realize that the indicators may be virtualized in some cases), and in those cases it should not attempt to indicate to the user via a user interface the state of the physical indicator which is controlled by a virtual indicator.

Virtual indicators are controlled by the OS for which they are generated, except that the platform may activate an OS' virtual Error Log indicator if the partition in which the OS resides abnormally terminates.

16.1.1.6 Service Focal Point (SFP) and Service Partition

The *Service Focal Point* (SFP), when it exists, will ultimately be the exclusive common point of control in the system for handling all service actions which are not resolved otherwise (for example, via Fault indicators). It interfaces with the error log where all the serviceable events are stored from the various OS and service processor diagnostics. The SFP, among other things, allows resetting of the Error Log light by the user, allows controlling the activation and deactivation of the FRU, connector, and Enclosure Identify indicators, and allows the clearing of the service actions in the error log.

The SFP shares access to some of the same indicators as one or more OS images, but needs access to the physical indicator state, and sometimes the state of all the virtual indicators for that physical indicator. If the SFP in a partitioned system were to be implemented on an OS image that runs non-trusted applications, then the SFP partition could not be given access to the physical and other OS' virtual service indicators, or covert storage channels would exist (see Section 16.1.1.5, "Covert Storage Channels," on page 516) between the SFP partition and the other OS partitions. This architecture assumes that the SFP is implemented as trusted or *privileged* entity which does not allow non-trusted applications running on the same OS image as the SFP, and therefore covert storage channels are not considered to exist between the SFP's privileged OS image and other OS images in the system.

The SFP may also be implemented on as a separate entity from the one being monitored. A system management entity like an HMC interfacing to the platform via firmware interfaces, or an external system management entity, are examples of such implementations.

For Lightpath Mode, the Triple-S UI is a user interface that is associated with a SFP. See Section 16.2.3, "Lightpath User Interface (UI) Requirements," on page 544.

The platform's physical indicators are accessible to the SFP through the normal indicator interface (LoPAPR indicator types 9006 and 9007).

16.1.1.7 Logical Indicators vs. Physical Indicators

A physical indicator is, in many cases, used to represent several logical and/or virtual indicators. For example, a physical FRU indicator can be used in Lightpath Mode to represent both a FRU Identify indicator and a FRU Fault indicator.

The hardware/firmware that implements the physical indicator's state machine is the entity which knows about the combining of the logical and virtual into the physical, and higher level software (OS and applications) that are given control of a logical or virtual indicator are only aware of the control of that logical or virtual indicator, and may not be even able to sense the state of the physical indicator (that is, can only sense the state of their logical or virtual ones).

The physical indicator state diagrams in Section 16.2.1.9, "Service Indicator State Diagrams," on page 533 indicate how logical and virtual indicators are merged into the physical ones. See also Section 16.1.1.5, "Covert Storage Channels," on page 516 relative to virtual indicators.

16.1.2 Machine Classes and Service Strategy

Two broad classifications of computer implementations are defined here for purposes of defining service indicator implementations. Table 198, “Machine Classifications and Service Characteristics,” on page 518 shows the comparison of these classes.

Table 198. Machine Classifications and Service Characteristics

Characteristic	Simple Class	Complex Class
Number of FRUs	Few	Many
Servicing performed by	Customer, generally	CE more than customer
Deferred maintenance	Very little	Enabled as much as possible[a]
Concurrent maintenance	Generally limited to redundant components (fans, power supplies) and I/O devices	Generally a higher level of concurrent maintenance
Value of a FRU Fault indicator	High	Questionable value due to complexity of the system
FRU Fault indicator implementation	Realistic	Complex, given the higher level of deferred and concurrent maintenance
Console interface	Rare	Standard
Platform service mode	Lightpath Mode platform	Guiding Light Mode platform or Lightpath Mode with Triple-S UI

a. Deferred maintenance is one of the big drivers towards use of Guiding Light mode or Lightpath Mode with Triple-S. That is, having many Fault indicators active at one time (FRUS waiting for service actions) can lead to confusion when service is being performed.

Determining whether a platform’s classification, and therefore the service mode of the platform is dependent on the product requirements, and is beyond the scope of this architecture, but might include:

- The RAS requirements for the platform. The considerations for this come from Table 198, “Machine Classifications and Service Characteristics,” on page 518.
- The mixture of machines expected in the environment. Although the Lightpath Mode and the Guiding Light Mode both contain the identify capability, and that could be considered to be the common denominator in servicing in a mixed environment, it could be that there are more Lightpath Mode platforms in the environment for which a new platform design is targeted, and therefore it might be desirable to make that new platform’s mode of operation be the Lightpath Mode for that reason.

16.1.3 General Information about Service Indicators

Indicators may serve multiple uses, but only as defined by this architecture. For example, a physical indicator used for a FRU is used for both the FRU Fault and FRU Identify indicators. Non-architected usages of an architected indicator are specifically disallowed by this architecture.

In some cases, an indicator may not be visible directly by the user without removing covers, components, etc. In this case, there is required to be one or more indicators that are higher in the hierarchy which get activated in conjunction with the target indicator. This functionality is called indicator *roll-up*. Due to the hierarchy, there might be multiple indicators that get rolled-up into a single indicator. The platform (not the OS) is responsible for indicator roll-up. An ex-

ample of a roll-up is that on the front of an enclosure[1] in a rack there is a summary LED that summarizes the Identify LEDs within or on the back of the enclosure in a rack, and then the multiple enclosure summary LEDs are summarized at the rack level with a light on the top of the rack. Another example of a roll-up indicator is the Enclosure Fault indicator on each enclosure in Lightpath Mode platforms, which summarizes the Fault indicators within the enclosure. These roll-up indicators are transparent to the OS, and sometimes to the firmware, with the exception that the enclosure level Identify indicators (or virtual versions thereof, in the case of partitioned systems) may be accessible to the OS via the 9007 indicator type. The indicators that are provided for roll-up from FRU to enclosure to rack are identified by this architecture. Platforms may have unique indicators which are not visible to the OS and which are not defined by this architecture. These will not share the same indicator as used by one of the indicators which is architected, including indicators in the roll-up path, except as explicitly allowed by the architectural requirements in this architecture. In addition to the roll-up to a higher level indicator for visibility, the platform may also provide duplicate indicators for some of the indicators. For example, there may be a front and rear indicator for the enclosure indicators. These duplicate indicators are not defined by this architecture except that as for roll-up indicators, the platform is responsible for controlling any duplicate indicators and for not making the duplicates visible to the controlling entities. Finally, FRU indicators are required to be visible to the user during a service action. This may require, for example, that the indicator be able to be lit after power is removed from the system, requiring storage of power on the component with the indicator (for example, via a capacitor) and activation of the indicators by a push button by the user.

An OS image is given access to the FRU Identify indicators when the OS image fully owns the resources, and is given access to the Enclosure Identify indicator for any enclosure in which the OS image fully owns any resource. An OS image is given access to the FRU Fault indicators when the OS image owns all or part of the FRU. The Enclosure Fault indicators are roll-up only indicators and access to these indicators are not given to the OS.

In a partitioned system (logical or physical), there may be several virtual Enclosure Identify indicators and one physical Enclosure Identify indicator. In this case, the OS images are only given access to their copy of the virtual Enclosure Identify indicator, and do not have direct access to the physical Enclosure Identify indicator. Activating any virtual Enclosure Identify indicator which is associated with an enclosure activates the physical one for that drawer (if not already activated). Turning off the last virtual Enclosure Identify indicator for an enclosure turns off the physical one for that enclosure, providing all other Identify indicators in the enclosure are also off. If software in a partition senses the state of the virtual Enclosure Identify indicator, it needs to take into consideration that it may be seeing the virtual state and not the real state of the indicator, with the virtual state being what the partition set the indicator to, and this is not necessarily what the physical indicator is actually displaying.

The Error Log indicator is located on the System Enclosure (the CEC enclosure) and is used to indicate that there was a failure in the system. This indicator may also be used by the system to indicate that some other attention is needed in the system. This Error Log indicator is the starting point for the determination of the necessary action by the user.

In a partitioned system (logical or physical), there may be several virtual Error Log indicators and one physical Error Log indicator. Activating a virtual Error Log indicator activates the physical one. Turning off the last virtual Error Log indicator turns off the physical one. If software in a partition senses the state of the Error Log indicator, it needs to take into consideration that it may be seeing the virtual state and not the real state of the indicator, with the virtual state being what the partition set the indicator to, and this is not necessarily what the physical indicator is actually displaying.

For Guiding Light Mode platforms, the FRU Identify indicators are the primary method for pointing to failing FRUs. For Lightpath Mode platforms, it is expected that the FRU Identify indicators will be used as a secondary assistance for FRU fault identification (the FRU Fault indicators being the primary). In both cases, the FRU Identify indicators can be used to assist with such things as identifying where an upgrade should be inserted.

The general rules for activation and deactivation of indicators can be found in Requirements R1–16.2.1.1–3 and R1–16.2.1.1–4, and more explicit requirements of individual indicators in the state diagrams in Section 16.2.1.9, “Service

1.Note that the enclosure is sometimes called the “unit,” but a unit is not necessarily a drawer and a drawer is not necessarily a unit, so the term “unit” is not be used here. Also note that an enclosure might be a drawer in a rack or might be part of a drawer. For example, some I/O drawers consist of two separate and independent enclosures. So, sometimes there may be multiple enclosure indicators per rack drawer. See also Section 16.1.1.1, ““Enclosure”, Packaging, and Other Terminology,” on page 511.

Indicator State Diagrams," on page 533. When the Triple-S UI is implemented, see also Section 16.2.3, "Lightpath User Interface (UI) Requirements," on page 544.

This architecture assumes that the control of multiple users doing identify operations at the same time, is under procedural control, and is not handled or controlled in any way by this architecture, OS, or firmware.

For Guiding Light Mode platforms, if a FRU contains a Fault indicator, then the Fault indicator is transparent to the OS and control of the FRU-level Fault indicator is entirely up to the FRU or to some OS-transparent method. For example, some power supplies contain a Fault indicator that does not get reported directly to the system controlling entity and which is turned off by a button on the power supply which is pushed when the service is complete.

16.1.4 Secondary Light Panels

A secondary light panel may be used to house roll-up indicators as indicated in the "intermediate" level or "secondary" level indicators in Figure 15, "Representation of the Indicators -- Lightpath Mode Platform," on page 521 and Figure 16, "Representation of the Indicators -- Guiding Light Mode Platform," on page 522. These panels may also house other indicators which would otherwise not have a home (for example, an over-temperature indicator).

Secondary light panels indicators are not to be used as replacement for FRU-level indicators. However, if an indicator is not directly visible when the unit is placed into the service position (for example, blocked by covers, baffles, cables, etc.), then the secondary light panel is one implementation to get around this restriction (other implementations may exist, for example light pipes, etc.).

16.1.5 Group Identify Operation

In some systems it may be desirable to identify a set of enclosures as being part of a group. This is called a group identify operation and can be performed by activating the appropriate Enclosure Identify indicators.

For platform or systems that consist of multiple enclosures, it may be necessary to change the state of one enclosure before servicing another enclosure. For example, a system drawer (primary enclosure) may need to be powered down before servicing an I/O drawer (secondary enclosure). It may be useful in this case for the servicer to be able to identify the various enclosures that are linked. In such implementations, the enclosures should be designed with a method to activate the Group Identify function, with the "group" being all linked enclosures. One implementation of this is to put a pushbutton in proximity to the blue Enclosure Identify indicator, which is then used to activate the blue Enclosure Identify indicators of all connected enclosures, and subsequently to deactivate all of them. It is suggested that with this implementation of the Group Identify function, that this switch toggle the Group Identify function for this set of enclosures, with each push toggling the Group Identify function. If it takes awhile to activate all the blue Enclosure Identify indicators in the group, it may be useful to give the user feedback that the button has been pressed. One way to do this is to put the blue Enclosure Identify indicator next to the pushbutton into the blink state (momentarily) until all the other blue Enclosure Identify indicators in the group have been activated, and when that is complete, to put this indicator into the Identify state (on solid).

16.1.6 System-Level Diagrams

The following figures are conceptual diagrams showing indicator roll-ups:

- Figure 15, "Representation of the Indicators -- Lightpath Mode Platform," on page 521.
- Figure 16, "Representation of the Indicators -- Guiding Light Mode Platform," on page 522.
- Figure 17, "Representation of the Indicators -- Rack System," on page 523.

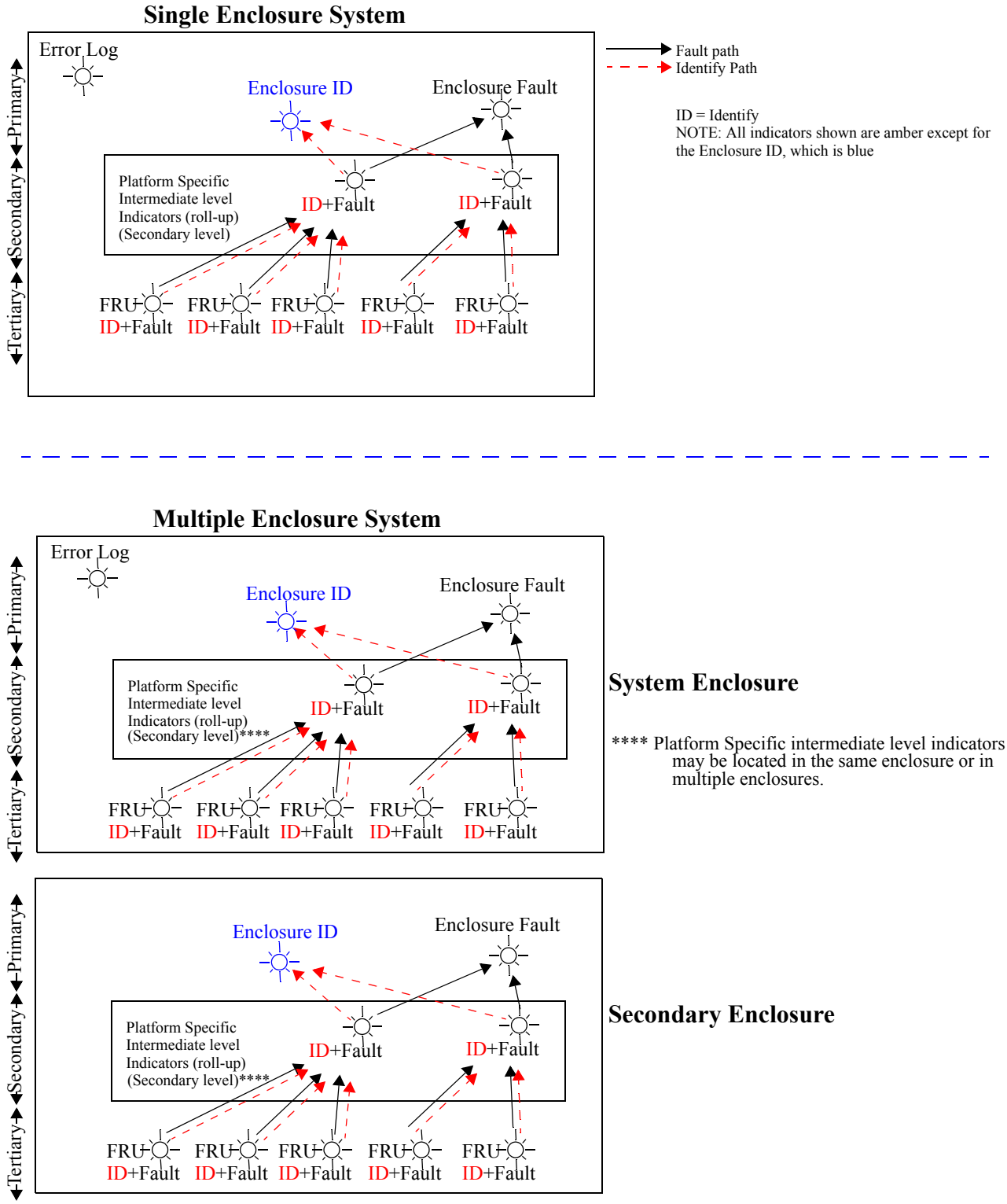


Figure 15. Representation of the Indicators -- Lightpath Mode Platform

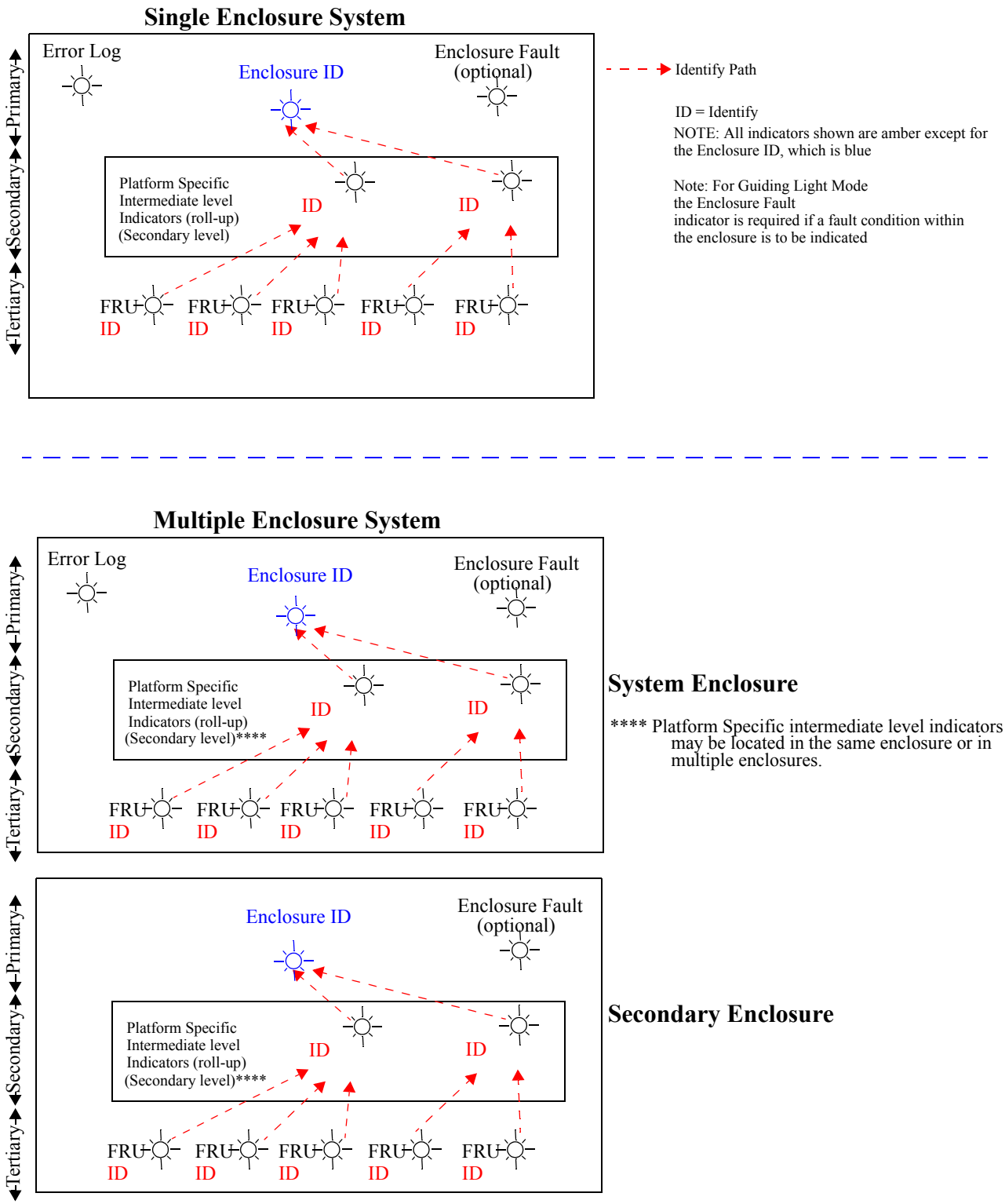
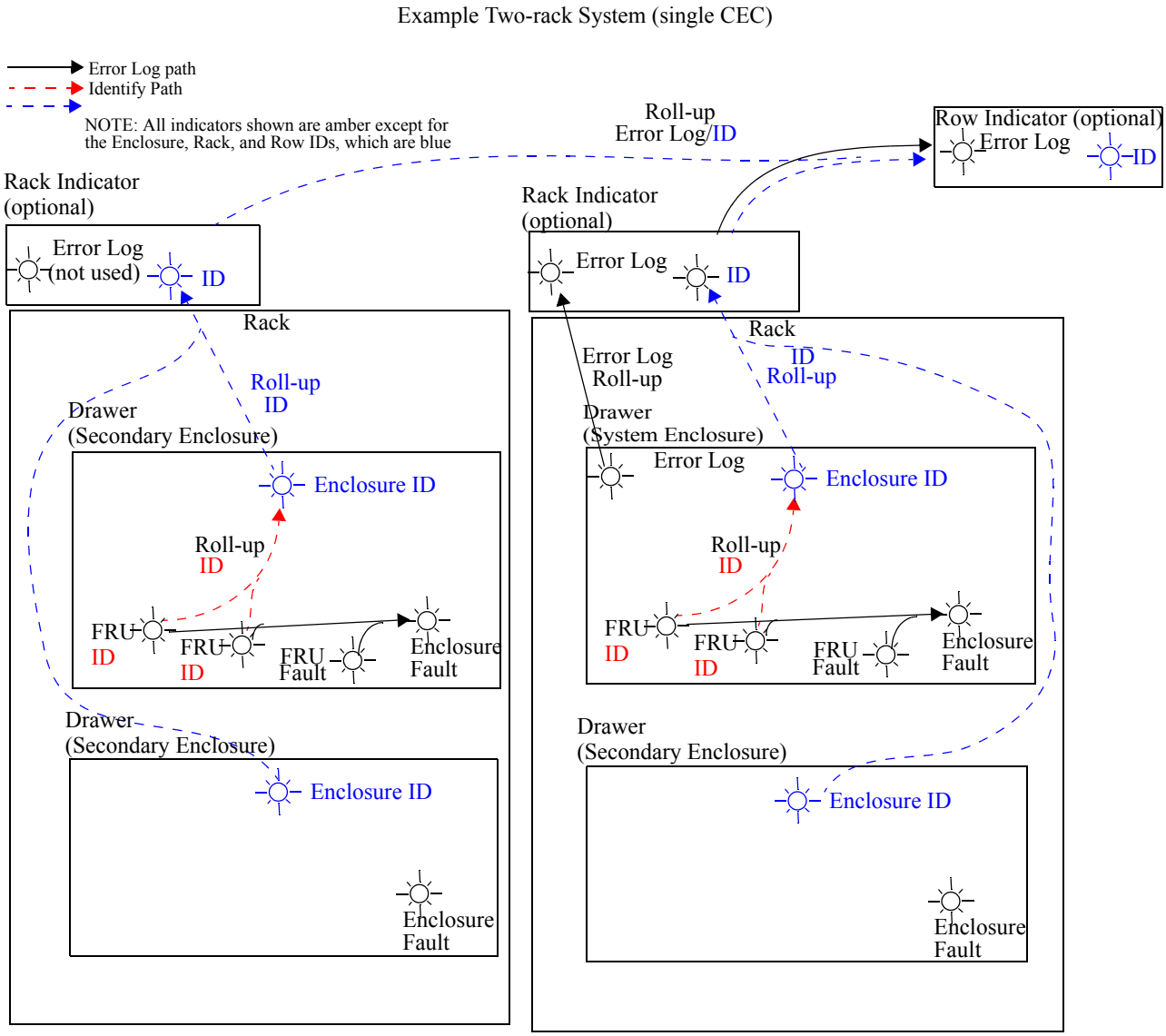


Figure 16. Representation of the Indicators -- Guiding Light Mode Platform



Notes: Guiding Light Mode platform shown, with optional Enclosure Fault indicators. For Lightpath Mode platforms, FRU Fault indicators would always exist and would roll up to the Enclosure Fault indicator, and additionally, the Rack and Row Identify indicators would have an additional Fault indicator (not shown) and the Fault indicators at the enclosure level would roll up to those.

Figure 17. Representation of the Indicators -- Rack System

16.2 Service Indicator Requirements

Service indicators are required on all platforms.

R1–16.2–1. All platforms must implement either the Lightpath Mode or the Guiding Light Mode of service indicators, and all components and enclosures (the primary enclosure and any secondary enclosures (for example, I/O expansion drawers)) within the platform must be operating in the same mode.

R1–16.2–2. Indicators defined by this architecture must not be used for any purpose other than is what is specified by this architecture, and only with the specific states defined by this architecture.

R1–16.2–3. All platforms must provide the **`"ibm,service-indicator-mode"`** property in the Open Firmware Device Tree root node.

16.2.1 Service Indicator General Requirements

This section details requirements of indicators that are not specifically LoPAPR indicator type 9006 or 9007 related. These are true even if the platform does not present any 9007 indicators to the OS. This includes requirements for platform actions like roll-up. For 9006 and 9007 specific requirements, see Section 16.2.2, “Requirements for 9002, 9006, and 9007 Indicators,” on page 543.

Requirements which are prefaced by “**For Lightpath Mode platforms:**” only apply to Lightpath Mode platforms. Requirements which are prefaced by “**For Guiding Light Mode platforms:**” only apply to Guiding Light Mode platforms. Requirements that are prefaced by neither, apply to both Lightpath Mode and Guiding Light Mode platforms. *Components* which are designed to work in both Lightpath Mode and Guiding Light Mode platforms, need to be able to comply with both Lightpath Mode and Guiding Light Mode sets of requirements, as well as the requirements that apply to all.

16.2.1.1 Fault Detection and Problem Determination Requirements

There are two general classifications of problems which are indicated by Service Indicators:

- An indication for FRUs that have failed and need to be replaced
- An indication of other system problems that may be causing performance degradation or which might cause failures in the future, for example:
  - A FRU that is predicted to fail (may be treated as a failing FRU by some implementations)
  - An over temperature conditions
  - A loss of redundancy that is not caused directly by a FRU failure (for example, greater than 100% of the power of the base power being used)
  - A configuration problem such as a missing resource, resource plugged into the wrong slot, or invalid configuration

The general model for use of the Error Log[1] and Enclosure Fault indicators is to indicate problems as follows:

- Activation of either the Error Log or Enclosure Fault indicators is accompanied by a log entry in an error log that can be queried by the user

1.Previously called the System Information (Attention) indicator.

- Activation of the Error Log indicator is used when the user needs to perform some procedure, or acknowledge some condition, prior to taking corrective action
  - For most types of problems, this requires the user to look into the error log at the start of the procedure
  - In some cases (generally for more common or more urgent problems), additional indicators may be provided by a system and activated to allow the user to determine the problem without looking into the error log (these additional indicators are generally not allowed to be the same indicators as defined by this architecture, except as allowed by this architecture)
  - The procedure performed by the user may include items like:
    - Activation of FRU Identify indicators (for example, as in Guiding Light Mode systems)
    - Removal and re-connection of cables, reseating of cards, etc.
- Activation of an Enclosure Fault (Lightpath Mode systems) is only allowed in the following cases:
  - As an indication of the roll-up of a FRU Fault indicator
  - In conjunction with a system error that prevents a FRU Fault indicator from being activated (this requires some other indication of the global failure problem, for example, an error code on an op panel)

The following requirements define the actions to be taken by a system on the detection of a fault.

R1–16.2.1.1–1. The detector of a fault condition must do the following:

- If the a fault occurs which cannot be isolated appropriately without the user performing some procedure, then activate the Error Log indicator.
- If a fault occurs which can be isolated to a single FRU and if there exists a Fault indicator for the FRU, then activate that FRU Fault indicator, otherwise activate the Error Log indicator.
- If a fault occurs which cannot be isolated to a single FRU and if there exists a Fault indicator for the most likely FRU in the FRU list, then activate that FRU Fault indicator, otherwise activate the Error Log indicator.
- If a fault occurs which is isolated to a group of FRUs (called a FRU group) and if there exists a Fault indicator for each of the FRUs, then activate all the FRU Fault indicators, otherwise activate the Error Log indicator.

See also, Appendix L, “When to use: Fault vs. Error Log Indicators (Lightpath Mode),” on page 885.

R1–16.2.1.1–2. *(Requirement Number Reserved For Compatibility)*

R1–16.2.1.1–3. Service Indicators (Error Log, Fault, and Identify) must be activated appropriately to guide a user to or through a service action or procedure.

R1–16.2.1.1–4. Service Indicators (Error Log, Fault, and Identify) must be deactivated appropriately, as follows:

a. A Service Indicator activated by an entity must be automatically deactivated by that entity when that entity can determine that the activation is no longer necessary, or

b. A Service Indicator must be automatically deactivated by the platform when the platform can determine that the activation is no longer necessary or may be necessary but will be redetected and therefore reactivated a reasonable time later, or

c. A Service Indicator must be automatically deactivated by a service procedure which fixes the issue that caused the indicator to be automatically activated in the first place, or

d. A Error Log or Identify indicator must be deactivated when a user request it to be deactivated by a system-level user interface.

e. For the Lightpath UI Base Enablement, as indicated in Requirement R1–16.2.3.1–1d and R1–16.2.3.1–2b.

R1–16.2.1.1–5. For each activation of the Error Log and Enclosure Fault Indicators, one of the following must be true:

a. If the platform is functional enough to allow it, then an associated entry must be made in an error log that can be queried by a user interface.

b. In the case where the platform is not functional enough to allow logging of an error log entry, then there must exist a way for the user to determine the failure associated with the indicator activation (for example, an error code on an op panel on the system).

Implementation Notes:

1. Requirement R1–16.2.1.1–4 are intentionally written general enough so that different platform types have some latitude in implementation of Service Indicators. However, see the state diagrams, Section 16.2.1.9, "Service Indicator State Diagrams," on page 533, for some explicit requirements for activation and deactivation of the various Service Indicators. Those state diagrams take precedence over Requirement R1–16.2.1.1–4. When the state diagrams and Requirement R1–16.2.1.1–4 do not give explicit direction for implementation, implementers should consider compatibility with existing implementations when making decisions about activation and deactivation.

2. In Requirement R1–16.2.1.1–4, the physical indicator may not be turned off when deactivated from an OS interface (versus a system-level interface), if another entity outside of that OS also has the physical indicator activated. That is, if the physical indicator is the combination of several logical indicators.

R1–16.2.1.1–6. The Error Log indicator must be activated only for Serviceable Events. Serviceable Events are platform, global, regional and local error events that require a service action and possibly a call home when the serviceable event must be handled by a service representative or at least reported to the service provider. Activation of the Error Log indicator notifies the customer of the event and the event indicates to the customer that there must be some intervention to rectify the problem. The intervention may be a service action that the customer can perform or it may require a service provider.

16.2.1.2 FRU-Level and Connector Indicator Requirements

The indicators specified in this section represent the lowest level indicators in the indicator roll-up hierarchy.

See also requirements in Section 16.2.1.7, "Additional Indicator Requirements," on page 531.

For the Lightpath UI, see also the requirements in Section 16.2.3, "Lightpath User Interface (UI) Requirements," on page 544.

R1–16.2.1.2–1. For Lightpath Mode platforms: All of the following must be true:

a. A FRU Fault indicator must be implemented for every replaceable FRU, with the states of "off" and "on," except for FRUs which are excepted in Requirement R1–16.2.1.2–4.

b. Clearing of the FRU Fault indicator from the Fault state must be the result of part of the repair action and must be transparent to the OS(s) and SFP (that is, the OS or SFP is not required to automatically clear a FRU Fault indicator).

c. The physical indicator which implements the FRU Fault indicator must also be Identify indicator and follow the requirements for Identify indicators.

R1–16.2.1.2–2. FRU indicators (Fault and Identify) must be visible to the user during a service action.

Implementation Note: Requirement R1–16.2.1.2–2 may require, for example, that the indicator be able to be lit after power is removed from the system, requiring storage of power on the component with the indicator (for example, via a capacitor) and activation of the indicators by a push button by the user (see Requirement R1–16.2.1.7–10 for requirements on this implementation). Another example would be via the use of a light pipe from the indicator to a visible place.

R1–16.2.1.2–3. For Guiding Light Mode platforms: If a FRU Fault indicator exists, then it must be transparent to the OS, SFP, and HMC and it must be independent of, and not physically combined into the same indicator with, any indicator defined by this architecture, including the setting of, resetting of, and displaying of the state of that indicator, except that a FRU Identify indicator may be activated to the Fault state (on solid) as a result of a FRU failure if all of the following are true:

a. The failure that is being indicated must be a failure which prevents the user from activating the said FRU Identify indicator to the Identify state.

b. Clearing of the FRU Fault indicator must be the result of part of the repair action and must be transparent to the OS, SFP, and HMC.

Architecture Notes:

1. For Guiding Light Mode platforms, the only FRU-level indicators that are allowed to be visible to an OS, SFP, or HMC are the FRU Identify indicators.

2. For Guiding Light Mode platforms, the only Fault indicator that is allowed to be visible to an OS, SFP, or HMC is the Error Log indicator.

3. Examples of the exception of the use of FRU and enclosure indicators in Requirement R1–16.2.1.2–3 as an indication of a fault are: when the path for controlling an Enclosure Identify indicator or FRU indicator in that enclosure is broken, or when the power supply in the enclosure is broken and the indicator cannot be activated to the Identify state. In these cases the FRU and/or enclosure indicators may be activated (transparently) to the Fault state to indicate the failure, and would be returned to the Normal state as a result of the repair action that fixes the problem.

R1–16.2.1.2–4. All platforms designs, except very low end servers,[1] must include an Identify indicator for every FRU with the states of "off" and "blink," except for the following classes of FRUs:

a. If a device driver has access to some standard form of Identify/Fault indicators for the DASD devices it controls (for example, some standard form of enclosure services), then the platform does not need to provide FRU indicators for these devices.

b. If a device driver has access to some standard form of Identify/Fault indicators for the removable media devices it controls (for example, some standard form of enclosure services), then the platform does not need to provide FRU indicators for these devices.

c. External enclosures other than PCI expansion enclosures, and external devices (for example, keyboard, mice, tablets) that attach via cable to IOAs, do not require FRU indicators.

d. Cables that connect from IOAs to the devices defined in parts a, b, and c of this requirement do not require FRU indicators.

e. Internal cables, interposers, and other FRUs which do not contain active components do not require FRU indicators.

1.The term "very low end servers" is not explicitly defined here, but is used to refer to implementations where FRU-level indicators cannot reasonably be implemented (for example, due to size constraints) or where the product can show explicit financial justification for not implementing.

Implementation Note: Even though an item falls into the list of possible exceptions in Requirement R1–16.2.1.2–4, the designer of such a component should verify that leaving off the FRU Identify indicator from their component will not prevent the systems in which that component is used from meeting their serviceability requirements.

R1–16.2.1.2–5. All FRU-level Identify indicators must implement the state diagram shown in Figure 18, "FRU or Connector Fault/Identify Indicator State Diagram," on page 535, except that the Fault state is not required for Guiding Light Mode platforms.

R1–16.2.1.2–6. All platforms must include an Identify indicator with the states of "off" and "blink" for every connector that is to be involved in an Identify operation.

R1–16.2.1.2–7. FRU-level and connector-level indicators must be made visible to the OS(s) as follows, and must be made transparent otherwise:

a. For Lightpath Mode platforms: FRU Fault indicators must be made visible to the OS for FRUs for which the OS image is expected to detect errors for either the entire FRU or part of the FRU.

b. FRU Identify indicators must be made visible to the OS for FRUs that are fully owned by the OS image.

c. Connector Identify indicators must be made visible to the OS for connectors that are fully owned by the OS image and for which a connector Identify indicator exists.

Implementation Notes:

1. In Requirement R1–16.2.1.2–7a, for FRU Fault indicators that are shared, the FRU Fault indicator is virtualized, so that one partition cannot view the setting by another partition, which would allow a covert storage channel (see also Section 16.1.1.5, "Covert Storage Channels," on page 516 and Section 16.2.1.6, "Shared Indicator (Multiple Partition System) Requirements," on page 531).

2. Ownership of a FRU or connector by the OS image is defined as being the condition of the FRU or connector being under software control by the OS, a device driver associated with the OS, or application software running on the OS.

R1–16.2.1.2–8. An OS which activates a FRU Identify indicator must provide a method of deactivating that indicator.

R1–16.2.1.2–9. *(Requirement Number Reserved For Compatibility)*

16.2.1.3 Enclosure-Level Indicator Requirements

See also requirements in Section 16.2.1.7, "Additional Indicator Requirements," on page 531

For the Lightpath UI, see also the requirements in Section 16.2.3, "Lightpath User Interface (UI) Requirements," on page 544.

R1–16.2.1.3–1. On the System Enclosure: The platform must implement an Error Log indicator and all of the following must be true:

a. The states of "off" and "on" must be implemented and must be used for the Error Log function

b. *(Requirement Number Reserved For Compatibility)*

c. This indicator must roll-up to the rack indicator, when the rack indicator is implemented, and for blade implementations, to the Chassis Error Log indicator.

d. The indicator must implement the state diagram shown in Figure 19, "Error Log Indicator State Diagram," on page 536.

e. The platform must provide a mechanism to allow the user to put the Error Log indicator into the off state.

R1–16.2.1.3–2. Except for enclosures that contain only FRUs that are exempted from FRU-level indicators as specified by Requirement R1–16.2.1.2–4 parts a, b, and c, and which also do not have any Connector Identify indicators, the platform must implement an Enclosure Identify indicator on all enclosures, and all the following must be true:

a. The states of "off," "blink," and "on" must be implemented and must be used for the Identify function.

b. This indicator must roll-up to the rack indicator, when the rack indicator is implemented, and for blade implementations, to the Chassis Enclosure Identify indicator.

c. The indicator must implement the state diagrams shown in Figure 20, "Enclosure Identify Indicator State Diagram for Scalable Systems," on page 537 and Figure 21, "Enclosure Identify Indicator State Diagram," on page 538.

R1–16.2.1.3–3. For Lightpath Mode Platforms: All the following must be true for the Enclosure Fault indicator:

a. The platform must implement an Enclosure Fault indicator on each enclosure in which there exists at least one FRU Fault indicator.

b. These indicators must implement the states of "off," "on," and "blip".

c. These indicators must implement the state diagram as shown in Figure 22, "Enclosure Fault Indicator State Diagram," on page 539.

d. These indicators must not be visible to any OS image.

e. The platform must provide a mechanism to allow the user to put each Enclosure Fault indicator into the blip state.

f. This indicator must roll-up to the rack indicator, when the rack indicator is implemented, and for blade implementations, to the Chassis Enclosure Fault indicator.

Implementation Note: One way of achieving Requirement R1–16.2.1.3–3e is to provide a pushbutton (for example, on the secondary indicator panel).

R1–16.2.1.3–4. ***(Requirement Number Reserved For Compatibility)***

R1–16.2.1.3–5. ***(Requirement Number Reserved For Compatibility)***

R1–16.2.1.3–6. Enclosure-level indicators must be made visible to the OS(s) as follows, and must be made transparent otherwise:

a. Enclosure Identify indicators must be made visible to the OS when the OS fully owns a FRU in the enclosure.

b. The Error Log indicator must be made visible to each OS image.

Implementation Notes:

1. In Requirement R1–16.2.1.3–6, for indicators that are shared, the indicator is virtualized, so that one partition cannot view the setting by another partition, which would allow a covert storage channel (see also Section 16.1.1.5, "Covert Storage Channels," on page 516 and Section 16.2.1.6, "Shared Indicator (Multiple Partition System) Requirements," on page 531).

2. Ownership of a FRU by the OS image is defined as being the condition of the FRU being under software control by the OS, a device driver associated with the OS, or application software running on the OS.

R1–16.2.1.3–7. An OS which activates an Error Log indicator must provide a method of deactivating that indicator, when such an activation is not be deactivated automatically as part of the service action.

Implementation Note: Relative to Requirement R1–16.2.1.3–7, it is recommended that an OS that activates an Error Log indicator, provide a way to deactivate that indicator, regardless of whether that indicator would be reset as part of a service action.

R1–16.2.1.3–8. An OS which activates an Enclosure Identify indicator must provide a method of deactivating that indicator.

R1–16.2.1.3–9. ***(Requirement Number Reserved For Compatibility)***

R1–16.2.1.3–10. For Guiding Light Mode Platforms: If a FRU Fault indicator exists, then it must not roll-up to the Enclosure Identify or Error Log indicator, and if there is such a requirement to roll-up such an indicator, then the enclosure must implement an Enclosure Fault indicator, with the same requirements as the Enclosure Fault indicator for Lightpath Mode platforms.

16.2.1.4 Rack-Level Indicator Requirements

See also requirements in Section 16.2.1.7, “Additional Indicator Requirements,” on page 531

R1–16.2.1.4–1. If a platform implements a rack-level indicator then all of the following must be true:

a. The rack indicator must be transparent to the OS, SFP, and HMC.

b. The rack indicator must be Highly visible[1] (distance and viewing angle) with covers in place.

c. For Lightpath Mode: The rack tower indicator must implement the state diagram indicated in Figure 26, “Rack-level Error Log Indicator State Diagram,” on page 542, Figure 27, “Rack-level Fault State Indicator Diagram,” on page 542, and Figure 28, “Rack-level Enclosure Identify Indicator State Diagram,” on page 542.

d. For the Guiding Light Mode: The rack tower indicator must implement the state diagram indicated in Figure 26, “Rack-level Error Log Indicator State Diagram,” on page 542, Figure 28, “Rack-level Enclosure Identify Indicator State Diagram,” on page 542, and if the optional Enclosure Fault indicators are implemented, then Figure 27, “Rack-level Fault State Indicator Diagram,” on page 542.

16.2.1.5 Row-Level Indicator Requirements

R1–16.2.1.5–1. If a system implements a row-level indicator to roll-up a row of rack-level indicators, then the following must be true for these indicators:

a. The indicator must be transparent to the OS, SFP, and HMC.

b. For Lightpath Mode: This indicator must implement the state diagram indicated in Figure 29, “Row-level Error Log State Diagram,” on page 543, Figure 30, “Row-level Fault State Diagram,” on page 543, and Figure 31, “Row-level Identify State Diagram,” on page 543.

c. For the Guiding Light Mode: This indicator must implement the state diagram indicated in Figure 29, “Row-level Error Log State Diagram,” on page 543, Figure 31, “Row-level Identify State Diagram,” on page 543, and if the optional Enclosure Fault indicators are implemented, then Figure 30, “Row-level Fault State Diagram,” on page 543.

1.As defined by our usability groups

16.2.1.6 Shared Indicator (Multiple Partition System) Requirements

To avoid covert storage channels (see Section 16.1.1.5, “Covert Storage Channels,” on page 516), virtual indicators are required for physical indicators which are shared between OS images.

R1–16.2.1.6–1. If a physical indicator (Fault or Identify) is shared between more than one partition, all the following must be true:

a. Except where there is explicit trust between the partitions, the platform must provide a separate virtual indicator to each non-trusted partition for each shared physical indicator and must control the physical indicator appropriately, as indicated in the state diagrams in Section 16.2.1.9, “Service Indicator State Diagrams,” on page 533.

b. If software in a partition senses the state of the virtual indicator, it must take into consideration that it is seeing the virtual state and not the real state of the indicator, with the virtual state being what the partition set the indicator to, and this is not necessarily what the physical indicator is actually displaying.

c. The SFP must be given access (sense and set) to the physical FRU level indicators, and the platform must clear all the corresponding virtual indicators when physical indicator is cleared by the SFP.

d. The SFP must be given access (sense and set) to the physical Error Log indicator, and the platform must not clear the corresponding virtual indicators when physical indicator is cleared by the SFP.

Architecture Note:

1. In Requirement R1–16.2.1.6–1, an example of “explicit trust” is where the sharing partitions are the SFP and one other partition, where the SFP is running in an OS where all the applications and drivers can be trusted to not open a covert channel to the other OS or application in that other partition.

2. In Requirement R1–16.2.1.6–1, it may be possible for the SFP to get direct access to the virtual indicators, but such access is beyond the scope of this architecture.

16.2.1.7 Additional Indicator Requirements

R1–16.2.1.7–1. A user interface which presents to a user the state of the Identify indicators or which allows the user to set the state of the Identify indicators, must be prepared for an indicator to disappear from the list of indicators available to the OS image (for example, a “no such indicator” response to a set request), and must provide the user with an appropriate message and recovery (for example, prompt the user whether they want to refresh the list of available indicators).

R1–16.2.1.7–2. The color of indicators must be as follows:

a. FRU Identify, FRU Fault, Enclosure Fault, Error Log indicators, and any roll-up indicators for Error Log (rack-level, blade system chassis-level, and row-level) must be amber.

b. The Enclosure Identify indicators and any roll-up indicators for these indicators (rack-level, blade system chassis-level, and row-level) must be blue.

R1–16.2.1.7–3. The blink rate of all Identify indicators which blink, must be nominally 2 Hz (minimum 1 Hz) with a nominal 50% duty cycle.

Implementation Note: The 1 Hz rate should not be used unless absolutely necessary. The 1 Hz rate is put in to be consistent with the industry standard SHPC specification, which specifies 2 Hz with 1 Hz minimum.

R1–16.2.1.7–4. The “blip” rate for the Enclosure Fault indicators when in the “remind” state must be nominally 0.5 Hz with a duty cycle of 0.2 seconds on, 1.8 seconds off.

R1–16.2.1.7–5. All indicator roll-up (activate and deactivate) must be controlled entirely by the platform and must be transparent to the OS, SFP, and HMC.

R1–16.2.1.7–6. Duplicate indicators that are implemented to reflect the same state as another indicator in the system (for example, an indicator on the back of an enclosure that is to reflect the same visible state as the enclosure indicator on the front of the enclosure) must be transparent to the OS and must be kept synchronized by the platform.

R1–16.2.1.7–7. The platform must provide a way to light all the indicators (Identify, Fault, Error Log, etc.) without any OS present, for test purposes (manufacturing, field service, etc.).

R1–16.2.1.7–8. The hardware must provide the firmware a way to read the state of the indicators (that is, the register which controls the visual state, not the actual visual state) as well as to set the state of the indicators.

R1–16.2.1.7–9. The platform must be designed such that permanently removing a FRU does not remove the capability to use the Identify indicator(s) remaining in the platform or affect any roll-up.

R1–16.2.1.7–10. In reference to Requirement R1–16.2.1.3–3e, if a capacitor and pushbutton are used to be able to activate the indicator after removal of the part, then all the following must be true:

a. For Lightpath Mode platforms: Both the Identify and the Fault states must be supported, and the indicator must activate when the push button is depressed and must go off (with the remaining capacitor charge maintained) when the pushbutton is released (Identify state is displayed as “blink” and Fault state as “on”).

b. For Guiding Light Mode platforms: The Identify state must be supported, and the indicator must activate (“blink”) when the push button is depressed and must go off (with the remaining capacitor charge maintained) when the pushbutton is released.

c. There must be a green indicator next to the pushbutton and the indicator must get turned on when the button is pressed and when there is charge in the capacitor, and must be off when the button is not pressed.

d. The capacitor must have the capability to store enough charge for two hours and after that period must be able to light for 30 seconds the green indicator and enough other indicators to be able to identify any necessary group of FRUs (for example, four additional indicators if a group of four DIMM locations is to be identified simultaneously).

Implementation Note: As part of Requirement R1–16.2.1.7–10, it is not necessary to roll-up any activated indicators to the next level when the button is pressed.

R1–16.2.1.7–11. All indicators which are under standby power must work the way they do when full power is applied to the system, including all of the following:

a. The indicators must continue to display the last state displayed when the system power went to standby-only power, unless the state is changed during the standby state by the user or by a service action.

b. The changing of the state to the Identify state and then back to the previous state by the user must be supported, when that functionality is supported during full system power.

Implementation Note: Internal to the platform firmware, it will most likely be required to have a common control point for all service indicators in order to meet the requirements and meet the necessary state information

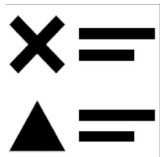
R1–16.2.1.7–12. Any secondary (intermediate) level roll-up indicator (see Section 16.1.4, “Secondary Light Panels,” on page 520) must behave as follows:

- Be blinking, if any Identify indicator that it represents is blinking
- Be on solid if any Fault indicator that it represents is on and no Identify indicator that it represents is blinking

- Be off if all indicators that it represents are off

R1–16.2.1.7–13. The icons used for the following indicators, and any roll-up of the same, must be as follows (see the usability specifications for size, color, and placement):

a. For the Error Log indicator:



b. For the Enclosure Fault indicator:

c. For the Enclosure Identify indicator:

16.2.1.8 Blade Systems Chassis-level Indicator Requirements

The following describes the chassis-level Error Log and Enclosure Identify indicator requirements for blade chassis implementations. These are basically the same as for the rack and row level indicators, except that the Enclosure Identify indicators are required to be able to be turned on/off by a user interface, unlike the rack and row level indicators.

For the Lightpath UI, see also the requirements in Section 16.2.3, “Lightpath User Interface (UI) Requirements,” on page 544.

R1–16.2–1. The blade chassis must implement an amber Error Log indicator, with the state diagram indicated in Figure 23, “For Blade Systems: Chassis-level Error Log Indicator State Diagram,” on page 540.

R1–16.2–2. The blade chassis must implement an amber Enclosure Fault indicator, with the state diagram indicated in Figure 24, “For Blade Systems: Chassis-level Fault Indicator State Diagram,” on page 540.

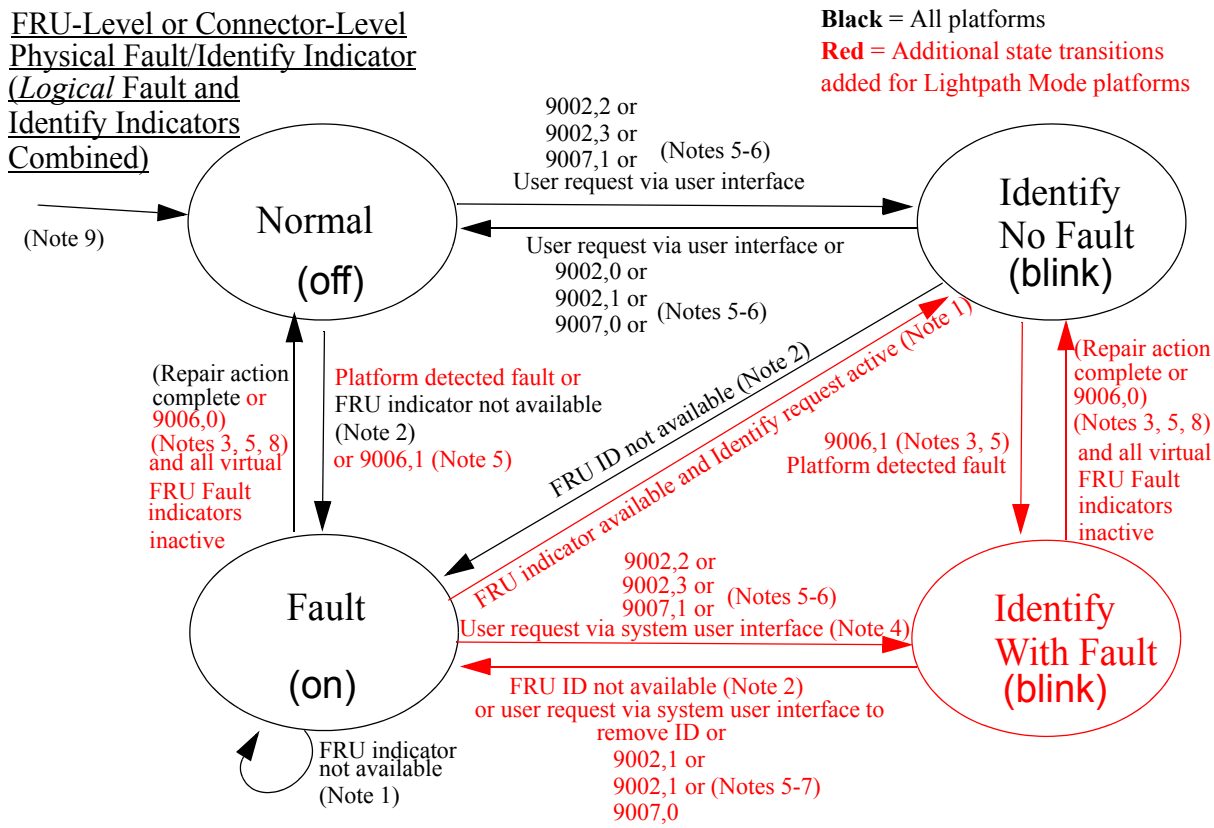
R1–16.2–3. The blade chassis must implement an blue Enclosure Identify indicator, with the state diagram indicated in Figure 25, “For Blade Systems: Chassis-level Enclosure Identify Indicator State Diagram,” on page 541.

16.2.1.9 Service Indicator State Diagrams

The following state diagrams show the transitions and states for the service indicators in the system.

Implementation Note: Activation of an indicator by a roll-up operation from a lower level indicator will prevent a user from turning off such an indicator from a user interface without turning off the lower level indicator. It is recommended, if possible, that in the user interface that allows the user to attempt to deactivate an indicator, provide to the user a message that the indicator cannot be deactivated, if attempted, when a roll-up to that indicator is active. That is, something better than just silently not turning off the indicator. Alternatively, the user can be

shown that the option of turning off such an indicator is not possible, when a roll-up to that indicator is active (for example, by graying out the option on the user interface).

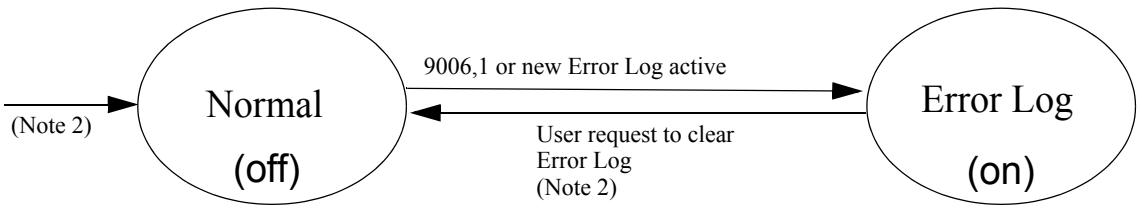


Notes:

1. Not being available means the failure that is being indicated must be a failure which prevents the user from activating the Identify for the FRU.
2. Transition to Fault state may occur if a failure occurs which would prevent the activation of the indicator into one of the Identify states. Not all FRU Fault indicators in an enclosure get activated like this simultaneously; only those that are directly involved with the fault (for example, like the FRU Fault indicator associated with the indicator controller hardware)
3. OS is not expected to change an indicator from Fault to Normal, but is permitted to do so (providing that it has access to the indicator because it owns the resource).
4. Transition from Fault to the Identify or Normal states by the OS may not be possible if a hardware fault causes a failure which prevents access to the indicator.
5. Format on the above diagram of "xxxx,y" means a call to the *set-indicator* or *ibm,set-dynamic-indicator* RTAS call with an indicator token of "xxxx" and a state value of "y" (only the token applicable for the specific indicator causes a state transition).
6. The 9002 Identify and Action are the same state.
7. 9006 FRU-level indicators only provided in Lightpath Mode platforms.
8. Fault indicators may be virtualized, with several OS images and firmware given access to a virtual FRU Fault indicator which controls the same physical Fault/Identify indicator. These get combined as shown in the state diagram, above; all virtual Fault indicators basically get ORed together.
9. This indicator may be forced to the Normal (off) state under certain circumstances (for example, see Requirement R1–16.2.1.1–4).
10. For the Lightpath UI, when implemented, other transition conditions are possible. See Section 16.2.3, "Lightpath User Interface (UI) Requirements," on page 544 for requirements.

Figure 18. FRU or Connector Fault/Identify Indicator State Diagram

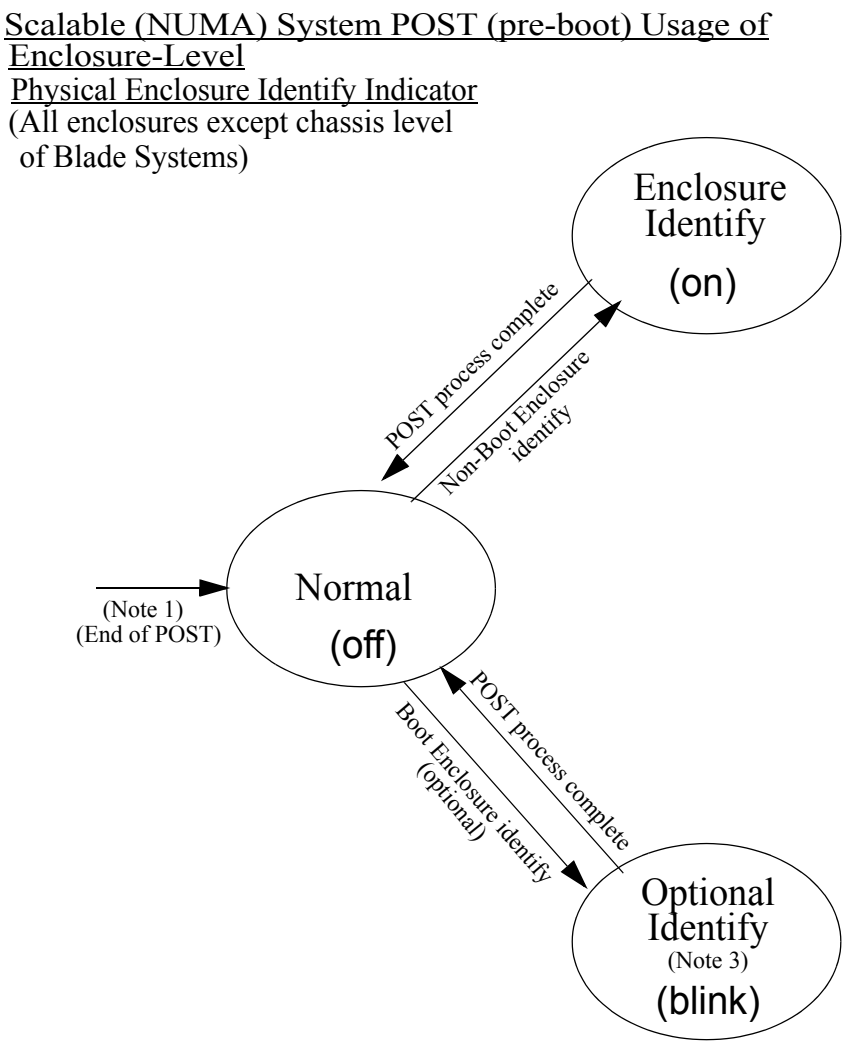
Enclosure-Level
Physical Error Log Indicator
(System Enclosure Only)



Notes:

1. Format on the above diagram of "xxxx,y" means a call to the *set-indicator* or *ibm,set-dynamic-indicator* RTAS call with an indicator token of "xxxx" and a state value of "y" (only the token applicable for the specific indicator causes a state transition).
2. This indicator may be forced to the Normal (off) state under certain circumstances (for example, see Requirement R1–16.2.1.1–4).
3. See Requirement R1–16.2.1.3–1e.
4. For the Lightpath UI, when implemented, other transition conditions are possible. See Section 16.2.3, "Lightpath User Interface (UI) Requirements," on page 544 for requirements.

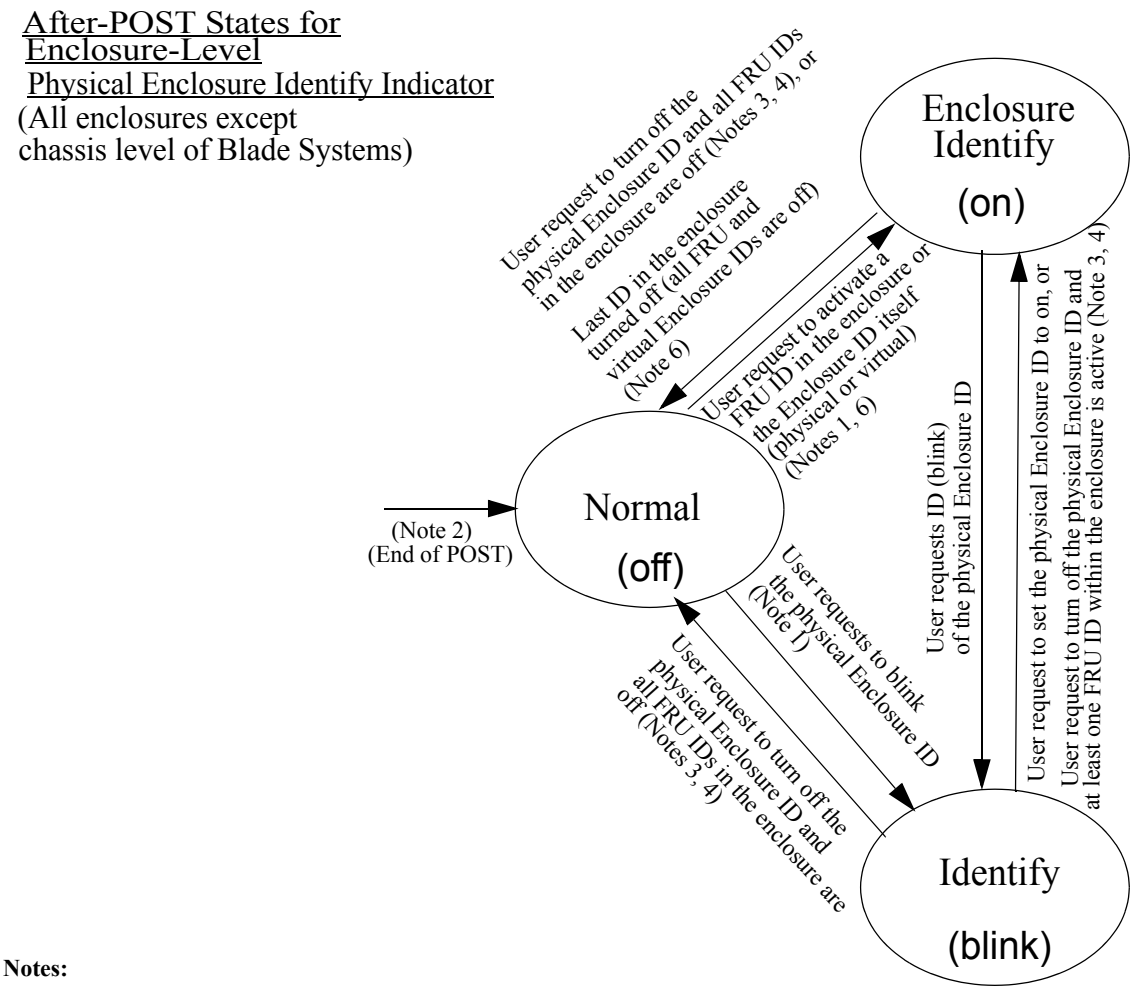
Figure 19. Error Log Indicator State Diagram



Notes:

1. This indicator may be forced to the Normal (off) state under certain circumstances (for example, see Requirement R1–16.2.1.1–4). This indicator is off at the end of POST.
2. The states in this diagram overlay the corresponding states in Figure 21, “Enclosure Identify Indicator State Diagram,” on page 538. This figure represents the POST states and Figure 21 the after-POST states.
3. The use of the Optional Identify state to indicate boot identify is only to be used for boot servers for scalable system nodes or blades of a blade system (for example, NUMA system nodes), and not for stand-alone systems or blades

Figure 20. Enclosure Identify Indicator State Diagram for Scalable Systems



Notes:

1. This indicator may be activated to the on state by any OS which is given access to the indicator per Requirements. For indicators that are shared by multiple OS instances, this indicator is virtualized (see Section 16.1.1.5, “Covert Storage Channels,” on page 516 and Section 16.2.1.6, “Shared Indicator (Multiple Partition System) Requirements,” on page 531). LoPAPR compliant OSs are only given the capability to activate the Enclosure ID to the on state, not to the blink state. The blink state may be activated through an external platform management interface by a user request through that interface to blink the physical Enclosure ID.

2. This indicator may be forced to the off state under certain circumstances (for example, see Requirement R1–16.2.1.1–4). This indicator is off at the end of POST.

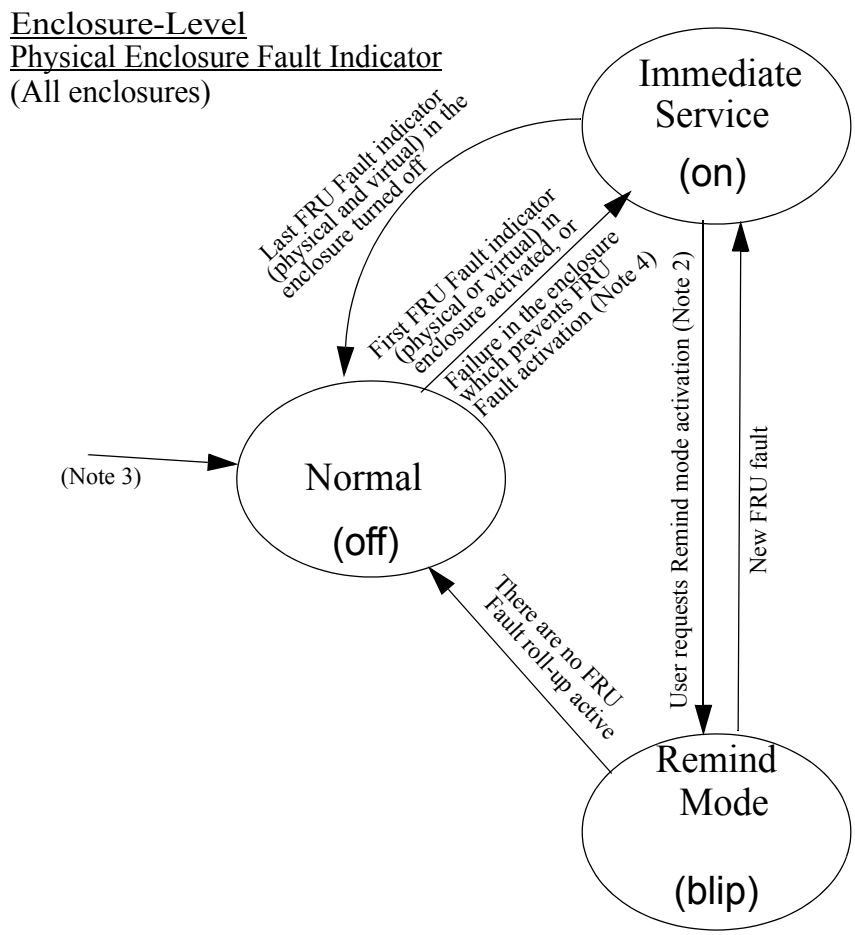
3. A user is not allowed to deactivate the Enclosure ID if there are still FRU IDs still active.

4. A user request through a privileged user interface (for example, via an SFP) to set the physical Enclosure ID to off, forces any virtual Enclosure ID indicators that are active (on or blink) to their off state, but this does not override any FRU ID roll-ups (see Note 3).

5. For Scalable (NUMA) systems, the states in this diagram overlay the corresponding states in Figure 20. This figure represents the after-POST states and Figure 20 the POST states.

6. A virtual Enclosure ID can be activated or turned off by the 9007 indicator token for the target Enclosure ID.

Figure 21. Enclosure Identify Indicator State Diagram



Note:

1. There is no direct activation or deactivation of this indicator by any OS.
2. See Requirement R1–16.2.1.3–3e and the Implementation Note below that requirement.
3. This indicator may be forced to the Normal (off) state under certain circumstances (for example, see Requirement R1–16.2.1.1–4).
4. Activation of an Enclosure Fault indicator without activating a FRU Fault indicator within the enclosure is to be used only in exceptional cases where the FRU Fault cannot be activated. In such cases the system is required to also provide further direction to the user on how to resolve the fault (for example, by providing an error code on an op panel on the system).

Figure 22. Enclosure Fault Indicator State Diagram

Chassis-Level
Physical Error Log
Indicator for Blade Systems

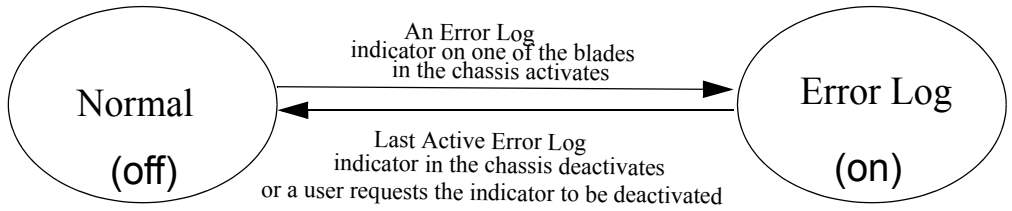


Figure 23. **For Blade Systems:** Chassis-level Error Log Indicator State Diagram

Chassis-Level
Physical Enclosure Fault
Indicator for Blade Systems

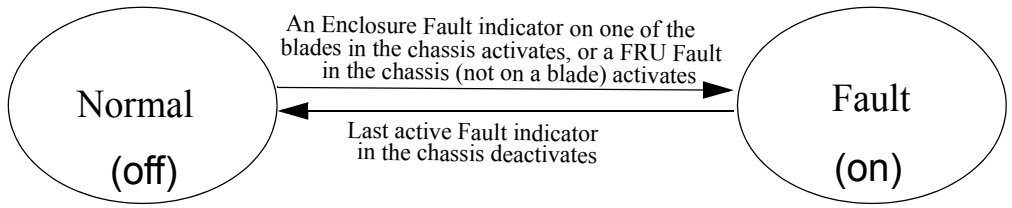
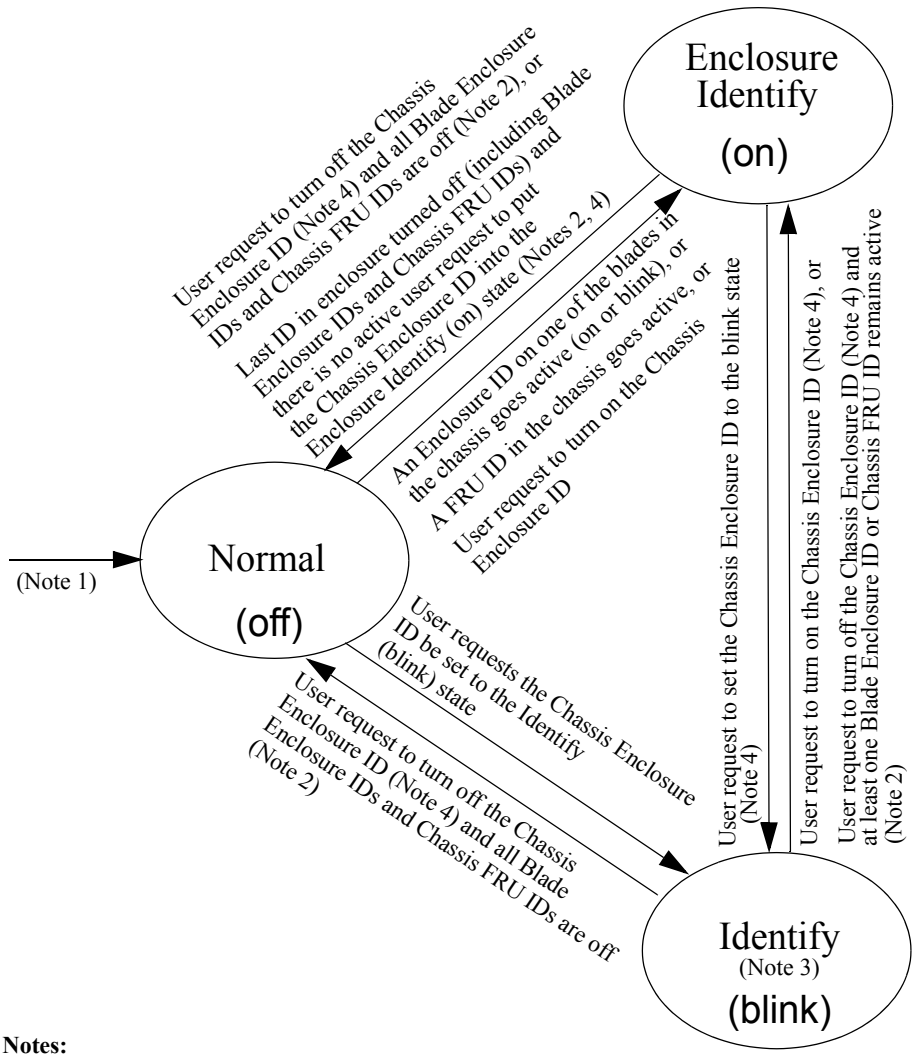


Figure 24. **For Blade Systems:** Chassis-level Fault Indicator State Diagram

Chassis-Level
Physical Enclosure Identify
Indicator for Blade Systems



Notes:

1. This indicator may be forced to the Normal (off) state under certain circumstances (for example, see Requirement R1–16.2.1.1–4).
2. A user is not allowed to deactivate the chassis Enclosure ID if there are still FRU Identify or Blade Enclosure Identify indicators still active (see state transition qualifiers in the above diagram).
3. A user request to set the Chassis Enclosure ID to the Identify (blink) state temporarily overrides roll-up operations (roll-up operations set the indicator to the on state).
4. A user request to change state of the Chassis Enclosure ID cancels any previous user request against the same indicator, replacing the user requested state with the new state.

Figure 25. **For Blade Systems:** Chassis-level Enclosure Identify Indicator State Diagram

Rack-Level
Physical Error Log
Indicator

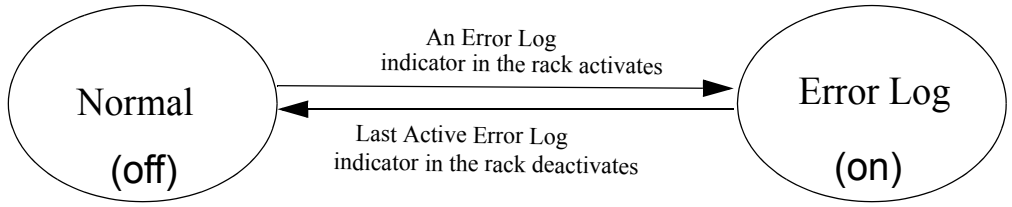


Figure 26. Rack-level Error Log Indicator State Diagram

Rack-Level
Physical Fault Indicator

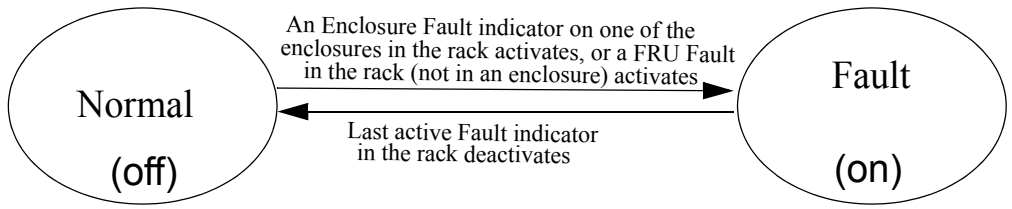


Figure 27. Rack-level Fault State Indicator Diagram

Rack-Level
Physical Identify Indicator

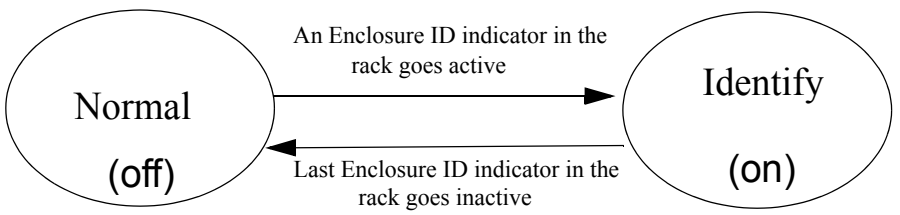


Figure 28. Rack-level Enclosure Identify Indicator State Diagram

Row-Level
Physical Error Log
Indicator

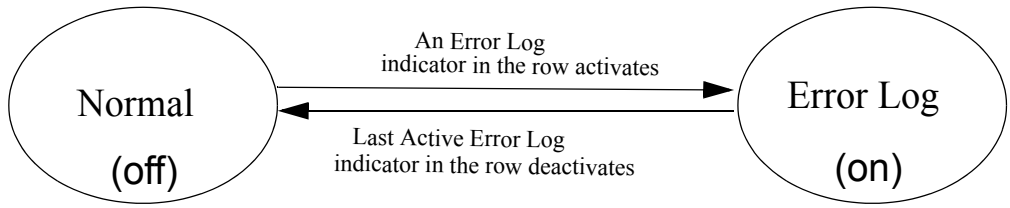


Figure 29. Row-level Error Log State Diagram

Row-Level
Physical Fault Indicator

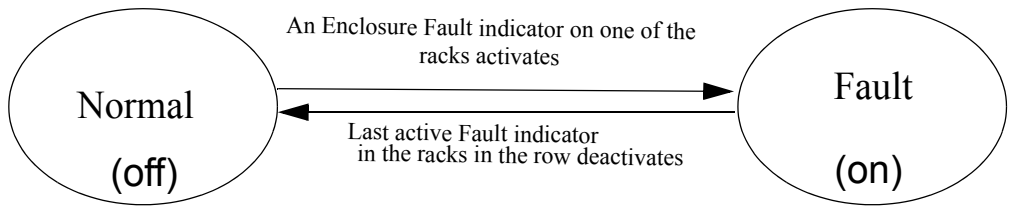
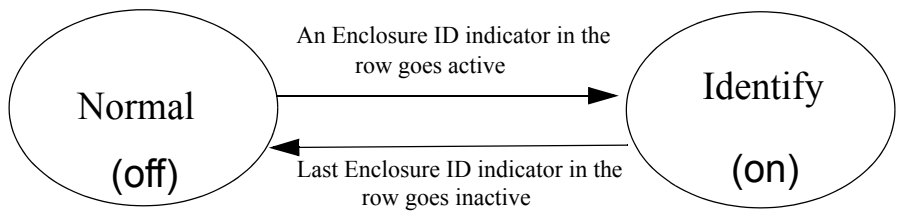


Figure 30. Row-level Fault State Diagram

Row-Level
Physical Identify Indicator



Note: A blinking Enclosure ID is assumed to be “active” for purposes of the Row Enclosure ID indicator state.

Figure 31. Row-level Identify State Diagram

16.2.2 Requirements for 9002, 9006, and 9007 Indicators

See Section 16.2.1, “Service Indicator General Requirements,” on page 524 for service indicator requirements that are not 9006 and 9007 specific.

R1–16.2.2–1. When the platform presents a 9006 indicator to an OS, the following must be true:

a. The platform must set the location code of the Error Log (9006) indicator and sensor to be the location code of the system and this indicator or sensor must be the first one in the list of 9006 indicators or sensors.

b. For Lightpath Mode platforms: The platform must set the location code of each FRU Fault indicator and sensor to be the location code of the component to which that indicator or sensor is associated.

c. For every 9006 indicator, there must be a corresponding 9006 sensor which has the same index as the corresponding indicator.

R1–16.2.2–2. When 9007 indicators are to be provided to an OS, the platform must implement the *ibm,get-indices* RTAS call and must present that call in the device tree for the OS, and the OS needing access to the 9007 indicators and sensors must use the *ibm,get-indices* call to get the indices of the 9007 indicators and sensors available to the partition at the time of the call.

Software Implementation Notes:

1. Relative to Requirement R1–16.2.2–2, due to Dynamic Reconfiguration, the indicators available at any point in time might be different than on a previous call to *ibm,get-indices*.

2. 9007 indicators may need to be provided to the OS in the order in which they are best displayed to the user, because the OS or the UI may not reorder them (for example, sort them) before presenting them to the user. This is true regardless of the method of presentation to the OS (OF device tree or *ibm,get-indices* RTAS call). Relative to presentation order, see also Requirement R1–16.2.2–3b.

R1–16.2.2–3. If a platform provides any 9007 indicators to the OS, then the following must be true:

a. The platform must set the location code of each Identify (9007) indicator and sensor (Enclosure, FRU, or connector) to be the location code of the enclosure, FRU, or connector to which that indicator is associated.

b. The System Enclosure Identify (9007) indicator must be the first indicator in the list of 9007 indicators.

c. For every 9007 indicator, there must be a corresponding 9007 sensor which has the same index as the corresponding indicator.

R1–16.2.2–4. A DR indicator (9002) must only be provided to an OS if that particular OS image owns that resource and is going to control the physical add, remove, and replace operations on the FRU which is pointed to by that particular DR indicator.

R1–16.2.2–5. If a PCI Hot Plug slot implements a single physical amber indicator for use as both the PCI Hot Plug DR indicator (for concurrent maintenance) and as the FRU Identify indicator, then that indicator must be presented to a LoPAPR compliant OS as both a 9002 and 9007 indicator.

R1–16.2.2–6. All platforms must provide the **`"ibm,fault-behavior"`** and **`"ibm,fru-9006-deactivate"`** properties in the **`root`** node of the OF device tree, both with a value of `1`.

16.2.3 Lightpath User Interface (UI) Requirements

The base Lightpath architecture does not provide a User Interface (UI), per se, when one considers a UI as being an interactive entity; that is, where the user can input requests as well as just see the faults. When enabling the Identify indicators of the Lightpath mode, a UI is necessary. This architecture will call this the Lightpath UI. The Lightpath UI is an interface between the Service Focal Point (SFP) and the user of the SFP, and at a minimum, provides an interface to show hidden Fault indicators (for example, see Section 16.2.3.2, “See/Select/Service (Triple-S) User Interface Requirements,” on page 547). A slightly more sophisticated Lightpath UI -- one with a General UI (GUI) such as an LCD or general display like provided by IBM Director -- is required to provide access to the Identify indicators.

Enablement of the Identify portion of Lightpath is important in larger systems for reasons of deferred maintenance and guided maintenance. In a system with deferred maintenance and Lightpath, many Fault indicators may remain lit, requiring directed repair via an Identify operation in order to see the component against which to do a particular repair action. In addition, guided maintenance may be required even if there is no failing component, to indicate to the user where to plug or un-plug components or cables.

When a Lightpath UI is available, the platform does not display logical Fault or Error Log on the physical indicators until a user requests such a display of the indicators, with the exception that the highest level roll-up indicators will be lit as a flag for the user to use the Lightpath UI to identify the problem. The request to display Fault and Error Log indicators may be made, for example, by pressing a button or series of buttons, or by checking a check-box on a more sophisticated Lightpath UI. The button(s) may be physical or may be on a device like an LCD panel or other Service Focal Point (SFP) display, like an IBM Director display.

Section 16.1.1.6, "Service Focal Point (SFP) and Service Partition," on page 517 defines an SFP as: "…common point of control in the system for handling all service actions which are not resolved otherwise (for example, via Fault indicators)." SFPs may or may not exist lower end systems, and may exhibit different levels of sophistication in larger systems. The following are some (not all) system implementation examples:

- For simple systems, there may be no SFP and no Lightpath UI, which means everything needs to be resolved by Fault indicators.
- For simple systems implementing Triple-S (see Section 16.2.3.2, "See/Select/Service (Triple-S) User Interface Requirements," on page 547), there may exist a simple SFP with a simple Lightpath UI like one or more physical push-buttons. This could be the System Error indicator with a physical button associated with it, with the SFP being firmware underlying the button to communicate with lower layers of firmware (for example, turn off FRU Fault indicators as they are activated by the firmware, turn on all active FRU Faults indicators on a button press). There may also be buttons for enabling the lower layers of the Fault indicator hierarchy, and these buttons inform the SFP firmware of the user's request to display Fault indicators on the physical indicators. In this case, the Lightpath UI is not full-function and does not provide for enablement of the Identify indicators. In this case, the firmware driving the Lightpath UI would use the Lightpath UI base enablement (see Section 16.2.3.1, "Lightpath UI Base Enablement Requirements," on page 545).
- For intermediate systems, the Lightpath UI could be an LCD panel. In this case, the firmware driving the Lightpath UI would use the Lightpath UI base enablement (see Section 16.2.3.1, "Lightpath UI Base Enablement Requirements," on page 545). The Triple-S UI is also possible (see Section 16.2.3.2, "See/Select/Service (Triple-S) User Interface Requirements," on page 547).
- For larger systems, the Lightpath UI could be part of a more sophisticated interface, like IBM Director. This more sophisticated interface would use the Lightpath UI base enablement (see Section 16.2.3.1, "Lightpath UI Base Enablement Requirements," on page 545). The Triple-S UI is also possible (see Section 16.2.3.2, "See/Select/Service (Triple-S) User Interface Requirements," on page 547).

16.2.3.1 Lightpath UI Base Enablement Requirements

This section defines the base enablement requirements for all Lightpath UI implementations. The Triple-S UI is one example of such a Lightpath UI that uses the Lightpath UI Base Enablement. Other Lightpath UIs are possible, and are not limited by this architecture.

R1–16.2.3.1–1. For the Lightpath UI Base Enablement: The platform must do all of the following:

a. Implement Lightpath Mode, as defined by this architecture, lighting FRU Fault indicators or Error Log indicator associated with a fault. Lightpath Mode includes the implementation of Identify indicators.

b. If the SFP is separate from the platform, then report to the SFP that the platform implements the Lightpath UI Base Enablement (explicitly or implicitly). (see implementation note, below)

c. Whenever possible, report all fault conditions which activate a FRU Fault indicator or Error Log indicator, up to the SFP, with enough information to allow determination by the SFP as to which FRU or Error Log indicators are activated and the possible failing FRU(s). See the implementation note, below, for the only exception cases allowed to this requirement.

d. Accept commands from the Service Focal Point (SFP) to put each indicator (FRU, Enclosure, etc.), into the Off, Fault and Identify states (that is, the SFP can control each indicator), and not report an activation or deactivation error to the SFP if the SFP requests putting the indicator into a state to which the indicator is already activated. See the implementation note, below, for the only exception cases allowed to this requirement.

e. Prevent multiple reports of same error, whenever possible. (see implementation note, below)

Implementation Notes:

1. Requirement R1–16.2.3.1–1b allows a SFP to manage multiple platforms that implement different Service Indicator modes. Note that this requirement can be implemented implicitly from other information reported to the SFP (for example, machine type/model).

2. In Requirement R1–16.2.3.1–1c and R1–16.2.3.1–1d, acceptable reasons for not being able to report errors to the SFP or have the SFP control the LEDs may include:

   - Loss of communications between the component and the SFP.

   - A fault indicator that is entirely controlled by an OS, hardware, or code, or an entity which is not in communications with the platform firmware or the SFP.

3. Requirement R1–16.2.3.1–1e prevents continual "blinking" of Fault indicator and the flooding of the SFP's event or error log.

R1–16.2.3.1–2. For the Lightpath UI Base Enablement: The Service Focal Point (SFP) must exist and must do all of the following:

a. Receive and log fault conditions reported by the platform. (see implementation note, below)

b. Turn off each Fault indicator or Error Log indicator associated with a fault condition, as soon as possible after the fault is reported, except as required to remain on by user request user request (for example, see Requirement R1–16.2.3.2–3).

c. Accept direction from a user to show any faults on the Fault and Error Log indicators (for example, see Requirement R1–16.2.3.2–3).

d. If the SFP contains a GUI (for example, an LCD display or a display like provided by IBM Director), accept direction from a user to Identify a FRU or connector for a service operation, and then turn off all activated FRU Fault and Error Log indicators, unless otherwise directed by the user (for example, by a check-box on the UI), and activate the FRU Identify (blink), along with the normal FRU roll-up defined by the base Lightpath Mode.

Implementation Notes:

1. Relative to Requirements R1–16.2.3.1–2a, a SFP may (but is not required to) do additional failure analysis, or may apply policy rules, on the failure(s) reported, and by doing so may change or re-prioritize the list of failures, such that the most likely failure(s) is (are) different than the fault indicator(s) that were initially turned on by the detecting entity. In which case, when the user requests that the indicators be reactivated, a different set may be activated than those that were originally activated.

2. In simpler systems, it is expected that there may only be one push-button implemented, and that would be associated with the highest level Fault roll-up indicator. For systems, or collection of systems that are managed by a SFP, which consist of many enclosures, it may be useful for an implementation to implement sev-

eral levels of buttons. For example, a SFP that manages multiple systems may (at least) implement one button per system.

16.2.3.2 See/Select/Service (Triple-S) User Interface Requirements

The Triple-S UI architecture is built on top of the Lightpath UI Base Enablement architecture, which is in turn built on top of the Lightpath architecture. The Triple-S architecture is basically defined as follows (see Requirements for specifics):

- Do not display Fault or Error Log on the physical indicators until user pushes a button or series of buttons.
  - Except that the highest level roll-up for the Enclosure Fault indicators and Error Log indicators will be activated if a lower level one of the same type was activated.
- After seeing the highest level roll-up for Enclosure Fault or Error Log being on, the user pushes one or more buttons (logical or physical) associated with those, and then the user *Sees* the Faults available for servicing.
- The user *Selects* the item they want to service, by observing the FRU Faults and selecting they want to then Service.
  - The *Selects* part of Triple-S may also involve activation of the FRU Identify indicator from the Lightpath UI.
- The user completes the Service action on that component which was selected.

R1–16.2.3.2–1. For the Triple-S UI: The Lightpath UI Base Enablement requirements must be met, as defined in Section 16.2.3.1, “Lightpath UI Base Enablement Requirements,” on page 545.

R1–16.2.3.2–2. For the Triple-S UI: The platform must provide one or more push-buttons (physical button, or logical on a GUI), each associated with a set of Fault indicators or Error Log indicators which is (are) to be used by the user to display (“show”) or not display (“hide”) fault conditions on those group of indicators, as defined by Requirement R1–16.2.3.2–3.

R1–16.2.3.2–3. For the Triple-S UI: The Service Focal Point (SFP) must accept direction from a user from a push-button (physical or logical) press to show any fault conditions on, or hide all fault conditions from, the physical indicators (FRU Faults and any associated roll-up indicators for those indicators), which are associated with the push-button (Fault or Error Log indicators). The fault conditions must represent any open problems known by the SFP related to the set of indicators associated with the push-button. The push of the button must be a toggle operation, with each press either going from the show state to the hide state or from the hide state to the show state, based on the state prior to the push-button press. The platform must turn off any indicators turned on by these push-button activations after a pre-set period of time after the button activation, unless the pre-set time is set to 0, in which case the indicators are left on until the button is press again or until the platform determines they are no longer needed to be on. (see implementation note, below).

R1–16.2.3.2–4. For the Triple-S UI: For more complex systems, and as determined by the RAS requirements for those systems, the SFP must implement a GUI (for example, an LCD or IBM Director display) and provide the capability to activate the Identify indicators, as defined in Requirement R1–16.2.3.1–2d.

Implementation Notes:

1. Relative to Requirement R1–16.2.3.2–3, a SFP may (but is not required to) do additional failure analysis, or may apply policy rules, on the failure(s) reported, and by doing so may change or re-prioritize the list of failures, such that the most likely failure(s) is (are) different than the fault indicator(s) that were initially turned on by the detecting entity. In which case, when the user requests that the indicators be reactivated, a different set may be activated than those that were originally activated.
2. Relative to Requirement R1–16.2.3.2–3, the set of indicators associated with a given push-button will normally be hierarchical, based on the FRU Fault or Error Log roll-up path. For example, if a push-button is associated with the Chassis Enclosure fault indictor, pressing that button would toggle the show/hide state for all fault indicators within that Chassis. Another example is pressing of a button associated with the System

Error roll-up indicator for a system, putting that system into the “show” state could put that system basically into a Lightpath (without Triple-S) mode, or “Lightpath Classic” mode. In this latter example, it is not quite like previous implementations of Lightpath because (1) service procedures may be different, (2) based on Implementation note (a) the set of FRU Faults activated by the SFP may be different than those activated by the entity detecting the error originally, and (3) the Identify function can be used.

16.3 Green Indicator Requirements

This chapter defines the platform requirements for green indicators.

The usage of green indicators has been separated from the rest of this chapter, because even though green indicators are used in some service procedures (for example, to check for presence or absence of power on a component or system), they are not to be used in lieu of amber FRU Fault and Identify indicators. That is, they should supplement, not replace, the amber indicators.

There are several exceptions to having all the green indicator requirements in this chapter:

- The green indicator associated with a capacitor and pushbutton implementation is specified in Requirement R1–16.2.1.7–10c.).
- The capability to light all green indicators, as well as the amber and blue indicators, for test purposes, is specified in Requirement R1–16.2.1.7–7.
- Unless indicated otherwise in this chapter, the blink rate for green indicators, when they blink, is specified in Requirement R1–16.2.1.7–3.

16.3.1 Green Indicator Uses and General Requirements

Green indicators generally are not used for indicating a fault condition.

R1–16.3.1–1. A green indicator must not be used in place of an amber Fault/Identify indicator, except when use of amber Fault/Identify indicator is not possible, and in this exceptional case, the green must be off or blinking to indicate the error condition.

Implementation Note: Examples where a green indicator might be used instead of an amber Fault/Identify indicator are:

- In a power supply to indicate lack of AC power (green off).
- In the case where there is insufficient power to power the component (green blinking).

R1–16.3.1–2. There must exist a green power indicator for every FRU that is to participate in concurrent maintenance (“hot plug” operation), unless that FRU does not require the removal of power to remove or insert that FRU.

16.3.2 Green Indicator States

This section attempts to capture the state requirements for all usages of green indicators. If a state or usage is not specified, then the user needs to get with the Architecture team for this architecture in order to add or replace any state or usage of that state.

16.3.2.1 Power Supply Green Indicators

R1–16.3.2.1–1. For power supply indicators, the platform must implement the states defined in Table 199, “Power Supply Green Indicator States and Usage,” on page 549 for each green indicator, and must use those states only for the usages stated in Table 199.

Table 199. Power Supply Green Indicator States and Usage

Green Indicator State	Usage
Any not already covered in this table	Consult with the xipSIA architecture team for the proper usage/behavior.
Off	◆ For the input power indicator: no input power. ◆ For the output power indicator: no output power.
On	◆ For the input power indicator: input power good. ◆ For the output power indicator: output power good.
Slow blink (1 Hz, 50% duty cycle)	Power supply (or supplies) are in the standby state. A power supply must not blink its green output power indicator unless that particular supply is in the standby state.

16.3.2.2 System Power Green Indicators

R1–16.3.2.2–1. For system power indicators, the platform must implement the states defined in Table 200, “System Power Green Indicator States and Usage,” on page 549 for each green indicator, and must use those states only for the usages stated in Table 200.

Table 200. System Power Green Indicator States and Usage

Green Indicator State	Usage
Any not already covered in this table	Consult with the xipSIA architecture team for the proper usage/behavior.
Off	System is off (no standby power).
On	System is on (operational state).
Fast blink (4 Hz, 50% duty cycle)	A determination is being made as to whether the system (for example, a Blade in a Blade System) has enough power available to it, in order to power up, or a determination has already been made that there is not enough power, and the indicator remains in this state.
Slow blink (1 Hz, 50% duty cycle)	System is in the standby power state.
Fade-in/fade-out cycling of the power LED as done in various PC and notebook manufacturers: the period of this fade-in/fade-out cycle is 2 seconds, gradually ranging from fully on to fully off	Systems that support system-level sleep states (such as the S3 sleep state) must use this state as a way to indicate the system is sleeping but still powered on.

16.3.2.3 HDD Green Indicators

R1–16.3.2.3–1. For Hard Disk Drives (HDD) the platform must implement the states defined in Table 201, “HDD Green Indicator States and Usage,” on page 550 for each green indicator, and must use those states only for the usages stated in Table 201.

Table 201. HDD Green Indicator States and Usage

Green Indicator State	Usage
Any not already covered in this table	Consult with the xipSIA architecture team for the proper usage/behavior.
Off	Platform specific.
On	Platform specific.
Flickering (randomly blinking)	HDD activity (HDD is powered on and is being used).

16.3.2.4 Other Component/FRU Green Indicators

This section attempts to capture the state requirements for usages of green indicators that are not specifically called out as special cases elsewhere in Section 16.3.2, “Green Indicator States,” on page 548. To reiterate what was specified, above: if a state or usage is not specified, then the user needs to get with the Architecture team for this architecture in order to add or replace any state or usage of that state.

R1–16.3.2.4–1. For FRUs or components other than specific ones specified elsewhere in Section 16.3.2, “Green Indicator States,” on page 548, which require power indicators, the platform must implement the states defined in Table 202, “Sub-Unit (Component) Green Indicator States and Usage,” on page 550 for each green indicator, and must use those states only for the usages stated in Table 202.

Table 202. Sub-Unit (Component) Green Indicator States and Usage

Green Indicator State	Usage
Any not already covered in this table	Consult with the xipSIA architecture team for the proper usage/behavior.
Off	Component/FRU is powered off and/or is not in operation.
On	Component/FRU is powered on.
Blink 1 Hz, 50% duty cycle	(Optional) Component/FRU is in transition to the off state. Note that although this is an optional state, it is highly recommended (for Human Factors reasons) for cases where it takes awhile to power off the component (for example, for hardware like a Blade in a Blade System that has to be quiesced before powering off).

16.3.2.5 Communication Link Green Indicators

R1–16.3.2.5–1. For communication links, the platform must implement the states defined in Table 203, “Communication Link Green Indicator States and Usage,” on page 550 for each green indicator, and must use those states only for the usages stated in Table 203.

Table 203. Communication Link Green Indicator States and Usage

Green Indicator State	Usage
Any not already covered in this table	Consult with the xipSIA architecture team for the proper usage/behavior.
Off	No link connection or link connected but no activity.
Flickering (randomly blinking)	Communication link activity.

16.4 Interpartition Logical LAN (ILLAN) Option

The Interpartition Logical LAN (ILLAN) option provides the functionality of IEEE VLAN between LPAR partitions. Partitions are configured to participate in the ILLAN. The participating partitions have one or more logical IOAs in their device tree.

The hypervisor emulates the functionality of an IEEE VLAN switch. That functionality is defined in IEEE 802.1Q. The following information on IEEE VLAN switch functionality is provided for informative reference only with the referenced document being normative. Logical Partitions may have one or more Logical LAN IOA's each of which appears to be connected to one and only one Logical LAN Switch port of the single Logical LAN Switch implemented by the hypervisor. Each Logical LAN Switch port is configured (by platform dependent means) as to whether the attached Logical LAN IOA supports IEEE VLAN headers or not, and the allowable VLAN numbers that the port may use (a single number if VLAN headers are not supported, an implementation dependent number if VLAN headers are supported). When a message arrives at a Logical LAN Switch port from a Logical LAN IOA, the hypervisor caches the message's source MAC address (2nd 6 bytes) to use as a filter for future messages to the IOA. Then the hypervisor processes the message differently depending upon whether the port is configured for IEEE VLAN headers, or not. If the port is configured for VLAN headers, the VLAN header (bytes offsets 12 and 13 in the message) is checked against the port's allowable VLAN list. If the message specified VLAN is not in the port's configuration, the message is dropped. Once the message passes the VLAN header check, it passes onto destination MAC address processing below. If the port is NOT configured for VLAN headers, the hypervisor (conceptually) inserts a two byte VLAN header (based upon the port's configured VLAN number) after byte offset 11 in the message.

Next, the destination MAC address (first 6 bytes of the message) is processed by searching the table of cached MAC addresses (built from messages received at Logical LAN Switch ports see above). If a match for the MAC address is not found and if there is no Trunk Adapter defined for the specified VLAN number, then the message is dropped, otherwise if a match for the MAC address is not found and if there is a Trunk Adapter defined for the specified VLAN number, then the message is passed on to the Trunk Adapter. If a MAC address match is found, then the associated switch port is configured and the allowable VLAN number table is scanned for a match to the VLAN number contained in the message's VLAN header. If a match is not found, the message is dropped. Next, the VLAN header configuration of the destination Switch Port is checked, and if the port is configured for VLAN headers, the message is delivered to the destination Logical LAN IOA including any inserted VLAN header. If the port is configured for no VLAN headers, the VLAN header is removed before being delivered to the destination Logical LAN IOA.

The Logical LAN IOA's device tree entry includes **`Unit Address`**, and **`"ibm,my-dma-window"`** properties. The **`"ibm,my-dma-window"`** property contains a LIOBN field that represents the RTCE table used by the Logical IOA. The Logical LAN hcall()s use the Unit Address field to imply the LIOBN and, therefore, the RTCE table to reference.

When the logical IOA is opened, the device driver registers, with the hypervisor, as the "Buffer List", a TCE mapped page of partition I/O mapped memory that contains the receive buffer descriptors. These receive buffers are mapped via a TCE mechanism from partition memory into contiguous I/O DMA space. The first descriptor in the buffer list page is that of the receive queue buffer. The rest of the descriptors are for a number of buffer pools organized by increasing size of receive buffer. The format of the descriptor is a 1 byte control field, 3 byte buffer length, followed by a 4 byte I/O address. The number of buffer pools is determined by the device driver (up to an architected maximum of 254). The control field in all unused descriptors is 0h00. The last 8 bytes are reserved for statistics.

When a new message is received by the logical IOA, the list of buffer pools is scanned starting from the second descriptor in the buffer list looking for the first available buffer that is equal to or greater than the received message. That buffer is removed from the pool, filled with the incoming message, and an entry is placed on the receive queue noting the buffer status, message length, starting data offset, and the buffer correlator.

The sender of a logical LAN message uses an hcall() that takes as parameters the Unit Address and a list of up to 6 buffer descriptors (length, starting I/O address pairs). The sending hcall(), after verifying the sender owns the Unit Address, correlates the Unit Address with its associated Logical LAN Switch port and copies the message from the send buffer(s) into a receive buffer, as described above, for each target logical LAN IOA that is a member of the specified

VLAN. If a given logical IOA does not have a suitable receive buffer, the message is dropped for that logical IOA (a return code indicates that one or more destinations did not receive a message allowing for a reliable datagram service).

The logical LAN facility uses the standard H_GET_TCE and H_PUT_TCE hcall()s to manage the I/O translations tables along with H_MIGRATE_DMA to aid in dynamic memory reconfiguration.

16.4.1 Logical LAN IOA Data Structures

The Logical LAN IOA defines certain data structures as described in following paragraphs. Figure 32, “Logical LAN IOA Structures,” on page 552 outlines the inter-relationships between several of these structures. Since multiple hcall()s as well as multiple partitions access the same structures, careful serialization is essential.

Implementation Note: During shutdown or migration of TCE mapped pages, implementations may choose to atomically maintain, within a single, two field variable, a usage count of processors currently sending data through the Logical LAN IOA combined with a quiesce request set to the processor that is requesting the quiesce (if no quiesce is requested, the value of this field is some reserved value). Then a protocol, such as the following, can manage the quiesce of Logical LAN DMA. A new sender atomically checks the DMA channel management variable -- spinning if the quiesce field is set and subsequently incrementing the usage count field when the quiesce variable is not set. The sender atomically decreases the use count when through with Logical Remote DMA copying. A quiesce requester, after atomically setting the quiesce field with its processor number (as in a lock), waits for the usage count to go to zero before proceeding.

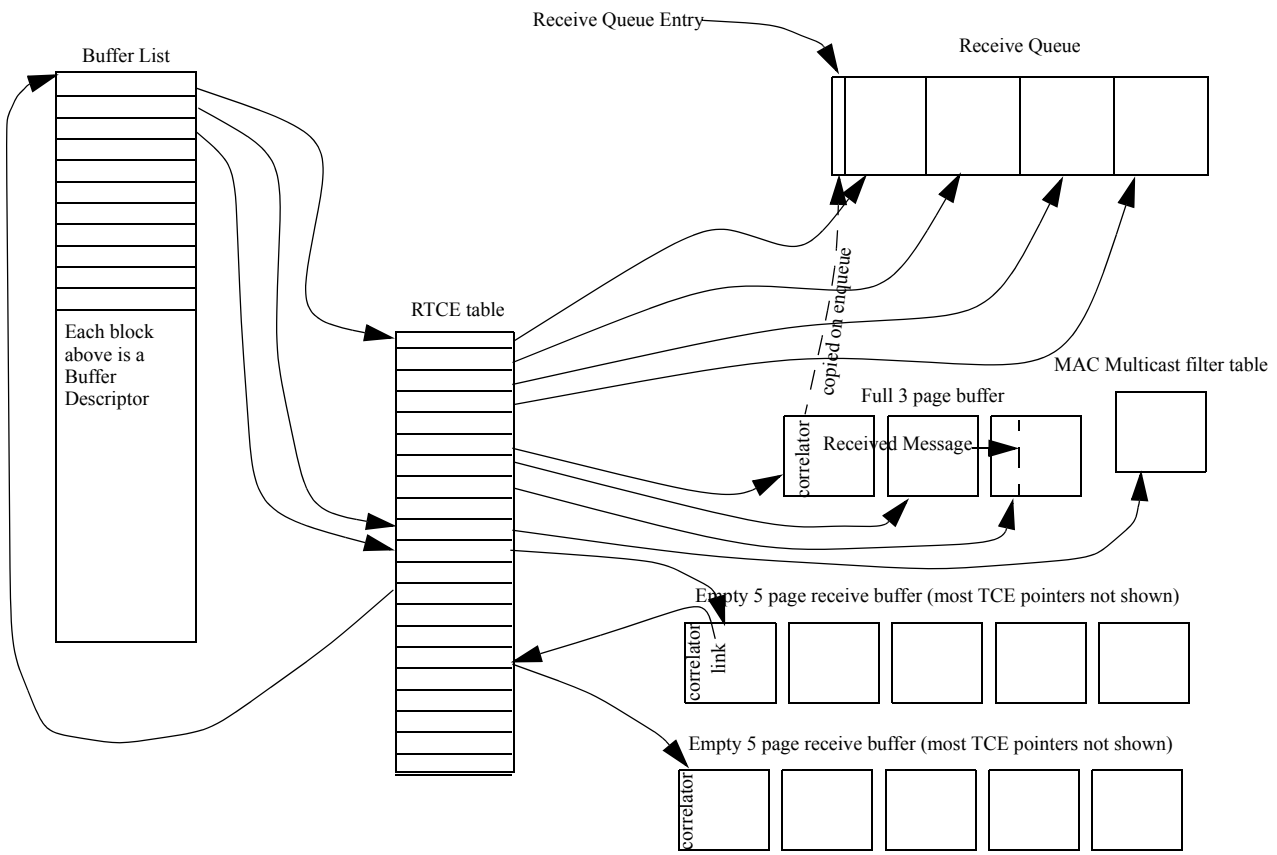


Figure 32. Logical LAN IOA Structures

16.4.1.1 Buffer Descriptor

The buffer descriptor is an 8 byte quantity, on an 8 byte boundary (so that it can be written atomically). The high order byte is control, the next 3 bytes consist of a length field of the buffer in bytes, the low order 4 bytes are a TCE mapped I/O address of the start of the buffer in I/O address space.

Bit 0 of the control field is the valid indicator, 0 means not valid and 1 is valid. Bits 2-5 are reserved.

Bit 1 is used in the receive queue descriptor as the valid toggle if the descriptor specifies the receive queue, else it is reserved. If the valid toggle is a 0, then the newly enqueued receive buffer descriptors have a valid bit value of 1, if the valid toggle is a 1, then the newly enqueued receive buffer descriptors have a valid bit value of 0. The hypervisor flips the value of the valid toggle bit each time it cycles from the bottom of the receive queue to the top.

Bit 6 is the No Checksum bit and indicates that there is no checksum in this packet. See Section 16.4.6.2, “ILLAN Checksum Offload Support Option,” on page 572 for more information on the usage of this bit.

Bit 7 is the Checksum Good bit and indicates that the checksum in this packet has already been verified. See Section 16.4.6.2, “ILLAN Checksum Offload Support Option,” on page 572 for more information on the usage of this bit.

16.4.1.2 Buffer List

This structure is used to record buffer descriptors of various types used by the Logical LAN IOA. Additionally, running statistics about the logical LAN adapter are maintained at the end of the structure. It consists of one 4 KB aligned TCE mapped page. By TCE mapping the page, the H_MIGRATE_DMA hcall() is capable of migrating this structure.

The first buffer descriptor (at offset 0) contains the buffer descriptor for the receive queue.

The second buffer descriptor (at offset 8) contains the buffer descriptor for the MAC multicast filter table.

It is the architectural intent that all subsequent buffer descriptors in the list head a pool of buffers of a given size. Further, it is the architectural intent that descriptors are ordered in increasing size of the buffers in their respective pools. The rest of the description of the ILLAN option is written assuming this intent. However, the contents of these descriptors are architecturally opaque, none of these descriptors are manipulated by code above the architected interfaces. This allows implementations to select the most appropriate serialization techniques for buffer enqueue/dequeue, migration, and buffer pool addition and subsequent garbage collection.

The final 8 bytes in the buffer list is a counter of frames dropped because there was not a buffer in the buffer list capable of holding the frame.

16.4.1.3 Receive Queue

The receive queue is a circular buffer used to store received message descriptors. The device driver sizes the buffer used for the receive queue in multiples of 16 bytes, starting on an 16 byte boundary (to allow atomic store operations) with, at least, one more 16 byte entry than the maximum number of possible outstanding receive buffers. Failure to have enough receive queue entries, may result in receive messages, and their buffers being lost since the logical IOA assumes that there are always empty receive queue elements and does not check. When the device driver registers the receive queue buffer, the buffer contents should be all zeros, this insures that the valid bits are all off.

If a message is received successfully, the next 16 byte area (starting with the area at offset 0 for the first message received after the registration of the receive queue and looping back to the top after the last area is used) in the receive queue is written with a message descriptor as shown in Table 204, “Receive Queue Entry,” on page 554. Either the entire entry is atomically written, or the write order is serialized such that the control field is globally visible after all other fields are visible.

Table 204. Receive Queue Entry

Field Name	Byte Offset	Length	Definition
Control	0	1	Bit 0 = the appropriate valid indicator. Bit 1 = 1 if the buffer contains a valid message. Bit 1 = 0 if the buffer does not contain a valid message, in which case the device driver recycles the buffer. Bits 2-5 Reserved. Bit 6: No Checksum bit. If a 1, then this indicates that there is no checksum in this packet (see Section 16.4.6.2, “ILLAN Checksum Offload Support Option,” on page 572 for more information on the usage of this bit). Bit 7: Checksum Good bit. If a 1, then this indicates that the checksum in this packet has already been verified (see Section 16.4.6.2, “ILLAN Checksum Offload Support Option,” on page 572 for more information on the usage of this bit).
Reserved	1	1	Reserved for future use.
Message Offset	2	2	The byte offset to the start of the received message. The minimum offset is 8 (to bypass the message correlator field); larger offsets may be used to allow for optimized data copy operations.
Message Length	4	4	The byte length of the received message.
Opaque handle	8	8	Copy of the first 8 bytes contained in the message buffer as passed by the device driver.

So that the device driver never has to write into the receive queue, the VLAN logical IOA alternates the value of the valid bit on each pass through the receive queue buffer. On the first pass following registration, the valid bit value is written as a 1, on the next as a zero, on the third as a 1, and so on. To allow the device driver to follow the state of the valid bit, the Logical LAN IOA maintains a valid bit toggle in bit 1 of the receive queue descriptor control byte. The Logical LAN IOA increments its enqueue pointer after each enqueue. If the pointer increment (modulo the buffer size) loops to the top, the valid toggle bit alternates state.

Following the write of the message descriptor, if enqueue interrupts are enabled and there is not an outstanding interrupt signaled from the Logical LAN IOA’s interrupt source number, an interrupt is signaled.

It is the architectural intent that the first 8 bytes of the buffer is a device driver supplied opaque handle that is copied into the receive queue entry. One possible format of the opaque handle is the OS effective address of the buffer control block that pre-ends the buffer as seen by the VLAN Logical IOA. Within this control block might be stored the total length of the buffer, the 8 byte buffer descriptor (used to enqueue this buffer using the H_ADD_LOGICAL_LAN_BUFFER hcall()) and other control fields as deemed necessary by the device driver.

When servicing the receive interrupt, it is the architectural intent that the device driver starts to search the receive queue using a device driver maintained receive queue service pointer (initially starting, after buffer registration, at the offset zero of the receive queue) servicing all receive queue entries with the appropriate valid bit, until reaching the first invalid receive queue entry. The receive queue service pointer is also post incremented, modulo the receive queue buffer length, and the device driver’s notion of valid bit state is also toggled/read from the receive queue descriptor’s valid bit toggle bit, on each cycle through the circular buffer. After all valid receive queue entries are serviced, the device driver resets the interrupt. See Section 16.4.3.9, “Other hcall()s extended or used by the Logical LAN Option,” on page 568. After the interrupt reset, the device driver again scans from the new value of the receive queue service pointer to pick up any entries that may have been enqueued during the interrupt reset window.

16.4.1.4 MAC Multicast Filter List

This one 4 KB page (aligned on a 4 KB boundary) opaque data structure is used by firmware to contain multicast filter MAC addresses. The table is initialized by firmware by the H_REGISTER_LOGICAL_LAN hcall(). Any modification of this table by the partition software (OS or device driver) is likely to corrupt its contents which may corrupt/affect the OS’s partition but not other partitions, that is, the hypervisor may not experience significant performance

degradation due to table corruption. However, for the partition that corrupted its filter list, the hypervisor may deliver multicast address packets that had previously been requested to be filtered out, or it may fail to deliver multicast address packets that had been requested to be delivered.

16.4.1.5 Receive Buffers

The Logical LAN firmware requires that the minimum size receive buffer is 16 bytes aligned on an 4 byte boundary so that stores of linkage pointer may be atomic. Minimum IP message sizes, and message padding areas force a larger minimum size buffer.

The first 8 bytes of the receive buffer are reserved for a device driver defined opaque handle that is written into the receive queue entry when the buffer is filled with a received message. Firmware never modifies the first 8 bytes of the receive buffer.

From the time of buffer registration via the H_ADD_LOGICAL_LAN_BUFFER hcall() until the buffer is posted onto the receive queue, the entire buffer other than the first 8 bytes are subject to modification by the firmware. Any modification of the buffer contents, during this time, by non-firmware code subjects receive data within the partition to corruption. However, any data corruption caused by errors in partition code does not escape the offending partition, except to the extent that the corruption involves the data in Logical LAN send buffers.

Provisions are incorporated in the receive buffer format for a beginning pad field to allow firmware to make use of data transfer hardware that may be alignment sensitive. While the contents of the Pad fields are undefined, firmware is not allowed to make visible to the receiver more data than was specifically included by the sender in the transfer message, so as to avoid a covert channel between the communicating partitions.

Table 205. Receive Buffer Format

Field Name	Byte Offset	Length	Definition
Opaque Handle	0	8	Per design of the device driver.
Pad 1	8	0-L1 cache line size	This field, containing undefined data, may be included by the firmware to align data for optimized transfers.
Message	defined by the “Message Offset” field of the Receive Queue Entry	12-N	The destination and source MAC address are at the first two 6 byte fields of the message, followed by the message payload.
Pad 2		To end of buffer	Buffer contents after the Message field are undefined.

16.4.2 Logical LAN Device Tree Node

The Logical LAN device tree node is a child of the **vdevice** node which itself is a child of **/**(the root node). There exists one such node for each logical LAN virtual IOA instance. Additionally, Logical LAN device tree nodes have associated packages such as obp-tftp and load method as appropriate to the specific virtual IOA configuration as would the node for a physical IOA of type network.

Logical IOA’s intrinsic MAC address -- This number is guaranteed to be unique within the scope of the Logical LAN.

Table 206. Properties of the Logical LAN OF Device Tree Node

Property Name	Required?	Definition
"name"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the virtual device name, the value shall be "l-lan".
"device_type"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the virtual device type, the value shall be "network".
"model"	NA	Property not present.
"compatible"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the programming models that are compatible with this virtual IOA, the value shall include "IBM,l-lan".
"used-by-rtas"	See definition column	Present if appropriate.
"ibm,loc-code"	Y	Property name specifying the unique and persistent location code associated with this virtual IOA, the value shall be of the form defined in Section 12.3.2.16, "Virtual Card Connector Location Codes," on page 335.
"reg"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the unit address (unit ID) associated with this virtual IOA presented as an encoded array as with **encode-phys** of length **"#address-cells"** value shall be 0xwhatever (virtual **"reg"** property used for unit address no actual locations used, therefore, the size field has zero cells (does not exist) as determined by the value of the **"#size-cells"** property).
"ibm,my-dma-window"	Y	Property name specifying the DMA window associated with this virtual IOA presented as an encoded array of three values (LIOBN, phys, size) encoded as with **encode-int, encode-phys,** and **encode-int.**
"local-mac-address"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the locally administered MAC addresses are denoted by having the low order two bits of the high order byte being 0b10.
"mac-address"	See definition column	Initial MAC address (may be changed by H_CHANGE_LOGICAL_LAN_MAC hcall()). Note: There have been requests for a globally unique mac address per logical LAN IOA. However, the combination of -- that requiring that the platform ship with an unbounded set of reserved globally unique addresses -- which clearly cannot work -- plus the availability of IP routing for external connectivity have overridden those requests.
"ibm,mac-address-filters"	Y	Property name specifying the number of non-broadcast multicast MAC filters supported by this implementation (between 0 and 255) presented as an encoded array encoded as with **encode-int**.
"interrupts"	Y	Standard property name specifying the interrupt source number and sense code associated with this virtual IOA presented as an encoded array of two cells encoded as with **encode-int** with the first cell containing the interrupt source number, and the second cell containing the sense code 0 indicating positive edge triggered. The interrupt source number being the value returned by the H_XIRR or H_IPOLL hcall().
"ibm,my-drc-index"	For DR	
"ibm,vserver"	Y	Property name specifying that this is a virtual server node.
"ibm,trunk-adapter"	See definition column	Property name specifying that this is a Trunk Adapter. This property must be provided when the node is a Trunk Adapter node.

Table 206. Properties of the Logical LAN OF Device Tree Node

Property Name	Required?	Definition
"ibm,illan-options"	See definition column	This property is required when any of the ILLAN sub-options are implemented (see Section 16.4.6, "Logical LAN Options," on page 571). The existence of this property indicates that the H_ILLAN_ATTRIBUTES hcall() is implemented, and that hcall() is then used to determine which ILLAN options are implemented.
"supported-network-types"	Y	Standard property name as per *Open Firmware Recommended Practice: Device Support Extensions* [5]. Reports possible types of "network" the device can support.
"chosen-network-type"	Y	Standard property name as per *Open Firmware Recommended Practice: Device Support Extensions* [5]. Reports the type of "network" this device is supporting.
"max-frame-size"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], to indicate maximum packet size.
"address-bits"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], to indicate network address length.
"ibm,#dma-size-cells"	See definition column	Property name, to define the package's dma address size format. The property value specifies the number of cells that are used to encode the size field of dma-window properties. This property is present when the dma address size format cannot be derived using the method described in the definition for the **"ibm,#dma-size-cells"** property in Appendix B, "LoPAPR Binding," on page 661.
"ibm,#dma-address-cells"	See definition column	Property name, to define the package's dma address format. The property value specifies the number of cells that are used to encode the physical address field of dma-window properties. This property is present when the dma address format cannot be derived using the method described in the definition for the **"ibm,#dma-address-cells"** property in Appendix B, "LoPAPR Binding," on page 661.

16.4.3 Logical LAN hcall()s

The receiver can set the virtual interrupt associated with its Receive Queue to one of two modes using the H_VIO_SIGNAL hcall(). These are:

1. Disabled (An enqueue interrupt is not signaled.)
2. Enabled (An enqueue interrupt is signaled on every enqueue)

Note: An enqueue is considered a pulse not a level. The pulse then sets the memory element within the emulated interrupt source controller. This allows the resetting of the interrupt condition by simply issuing the H_EOI hcall() as is done with the PCI MSI architecture rather than having to do an explicit interrupt reset as in the case with PCI LSI architecture.

The interrupt mechanism, however, is capable of presenting only one interrupt signal at a time from any given interrupt source. Therefore, no additional interrupts from a given source are ever signaled until the previous interrupt has been processed through to the issuance of an H_EOI hcall(). Specifically, even if the interrupt mode is enabled, the effect is to interrupt on an empty to non-empty transition of the queue.

16.4.3.1 H_REGISTER_LOGICAL_LAN

Syntax:

```
int64                              /* H_Success, H_Parameter, H_Hardware */
    hcall(const unit64 H_REGISTER_LOGICAL_LAN,/*Register structures for the logical LAN IOA*/
              uint64 unit-address,          /* As specified in the Logical LAN device tree node */
              uint64 buf-list,     /* I/O address of a 4 KB page (aligned) used to record registered input buffers */
              uint64 rec-queue,             /* Buffer descriptor of a receive queue */
```

```
        uint64 filter-list,          /* I/O address of a 4 KB page aligned broadcast MAC address filter list */
        uint64 mac-address);         /* The receive filter MAC address */
```

Parameters:

- unit-address: As specified in the Logical LAN device tree node **"reg"** property
- buf-list: I/O address of a 4 KB page (aligned) used to record registered input buffers
- rec-queue: Buffer descriptor of a receive queue, specifying a receive queue which is a multiple of 16 bytes in length and is 16 byte aligned
- filter-list: I/O address of a 4 KB page aligned broadcast MAC address filter list
- mac-address: The receive filter MAC address

Semantics:

- Validate the Unit Address else H_Parameter
- Validate the I/O addresses of the buf-list and filter-list is in the TCE and is 4K byte aligned else H_Parameter
- Validate the Buffer Descriptor of the receive queue buffer (I/O addresses for entire buffer length starting at the specified I/O address are translated by the RTCE table, length is a multiple of 16 bytes, and alignment is on a 16 byte boundary) else H_Parameter.
- Initialize the one page buffer list
- Enqueue the receive queue buffer (set valid toggle to 0).
- Initialize the hypervisor's receive queue enqueue pointer and length variables for the virtual IOA associated with the Unit Address. These variables are kept in terms of DMA addresses so that page migration works and any remapping of TCEs is effective.
- Disable receive queue interrupts.
- Record the low order 6 bytes of mac-address for filtering future incoming messages.
- Return H_Success.

16.4.3.2 H_FREE_LOGICAL_LAN

Syntax:

```
int64                                /* H_Success, H_Parameter, H_Hardware, H_Busy, */
                                     /* H_LongBusyOrder1mSec, H_LongBusyOrder10mSec */
    hcall(const uint64 H_FREE_LOGICAL_LAN,/*Deregister structures for the logical LAN IOA*/
          uint64 unit-address)       /* As specified in the Logical LAN device tree node */
```

Parameters:

- unit-address: Unit Address per device tree node **"reg"** property.

Semantics:

- Validate the Unit Address else H_Parameter
- Interlock/carefully manipulate tables so that H_SEND_LOGICAL_LAN performs safely.

- Clear the associated page buffer list, prevent further consumption of receive buffers and generation of receive interrupts.
- Return H_Success.

H_FREE_LOGICAL_LAN is the only valid mechanism to reclaim the memory pages registered via H_REGISTER_LOGICAL_LAN.

Implementation Note: If the hypervisor returns an H_Busy, H_LongBusyOrder1mSec, or H_LongBusyOrder10mSec, software must call H_FREE_LOGICAL_LAN again with the same parameters. Software may choose to treat H_LongBusyOrder1mSec and H_LongBusyOrder10mSec the same as H_Busy. The hypervisor, prior to returning H_Busy, H_LongBusyOrder1mSec, or H_LongBusyOrder10mSec, will have placed the virtual adapter in a state that will cause it to not accept any new work nor surface any new virtual interrupts (no new frames will arrive, etc.).

16.4.3.3 H_ADD_LOGICAL_LAN_BUFFER

Syntax:

```
int64                          /* H_Success, H_Parameter, H_Hardware */
    hcall(const uint64 H_ADD_LOGICAL_LAN_BUFFER,/* Adds a receive buffer to the receive buffer pool */
          uint64 unit-address,      /*As specified in the Logical LAN device tree node */
          uint64 buf)               /* Buffer descriptor of the receive buffer to add to the receive buffer pool */
```

Parameters:

- unit-address: Unit Address per device tree node **`"reg"`** property
- buf: Buffer Descriptor of new I/O buffer

Semantics:

- Checks that unit address is OK else H_Parameter.
- Checks that I/O Address is within range of DMA window.
- Scans the buffer list for a pool of buffers of the length specified in the Descriptor
- If one does not exist (and there is still room in the buffer list, create a new pool entry else H_Resource).
- Uses enqueue procedure that is compatible with H_SEND_LOGICAL_LAN hcall()'s dequeue procedure

Implementation Note: Since the buffer queue is based upon I/O addresses that are checked by H_SEND_LOGICAL_LAN, it is only necessary to insure that the enqueue/dequeue are internally consistent. If the owning OS corrupts his buffer descriptors or buffer queue pointers, this is caught by H_SEND_LOGICAL_LAN and/or the corruption is contained within the OS's partition.

Architecture Note: Consideration was given to define the enqueue algorithm and have the DD do the enqueue itself. However, no designs presented themselves that eliminated the timing windows caused by adding and removing pool lists without the introduction of OS/FW interlocks.

16.4.3.4 H_FREE_LOGICAL_LAN_BUFFER

Syntax:

```
int64                          /* H_Success, H_Parameter, H_Hardware, H_Not_Found*/
    hcall(const uint64 H_FREE_LOGICAL_LAN_BUFFER,/* Removes a buffer of the specified size, */
                               /*if available, from the receive buffer pool */
```

```
uint64 unit-address,/*As specified in the Logical LAN device tree node */
uint64 bufsize)     /* Size of the buffer to remove from the receive buffer pool */
```

Parameters:

- unit-address: Unit Address per device tree node **"reg"** property.
- bufsize: The size of the buffer that is being requested to be removed from the receive buffer pool.

Semantics:

- Check that unit address is valid, else return H_Parameter.
- Scan the buffer list for a pool of buffers of the length specified in bufsize, and return H_Not_Found if one does not exist.
- Place an entry on receive queue for buffer of specified size, with Control field Bit 1 set to 0, and return H_Success

16.4.3.5 H_SEND_LOGICAL_LAN

Syntax:

```
int64                       /* H_Success, H_Parameter, H_Dropped, H_Hardware, H_Busy*/
        hcall(const uint64 H_SEND_LOGICAL_LAN,/* Send a message on the logical LAN */
        uint64  unit-address,     /* As specified in the Logical LAN device tree node */
        uint64  buff-1,           /* The message content including source and destination MAC addresses */
        uint64  buff-2,           /* and optional IEEE VLAN header is contained in 1 - 6 buffers */
        uint64  buff-3,           /* each contiguous (in I/O space) buffer is passed using a buffer */
        uint64  buff-4,           /* descriptor. A buffer descriptor with a control byte of "invalid" */
        uint64  buff-5,           /* indicates the end of the message (only the number of buffers needed */
        uint64  buff-6,           /* are used). */
        uint64  continue-token)   /* value of 0 on first call, value returned in R4 on H_Busy*/
```

The H_Dropped return code indicates to the sender that one or more intended receivers did not receive the message.

Parameters:

- unit-address: Unit Address per device tree node **"reg"** property
- buff-1: Buffer Descriptor #1
- buff-2: Buffer Descriptor #2
- buff-3: Buffer Descriptor #3
- buff-4: Buffer Descriptor #4
- buff-5: Buffer Descriptor #5
- buff-6: Buffer Descriptor #6
- continue-token: Used to continue a transfer if H_Busy is returned. Set to 0 on the first call. If H_Busy is returned, then call again but use the value returned in R4 from the previous call as the value of continue-token.

Semantics:

- If continue-token is non-zero, then do appropriate checks to see that parameters and buffers are still valid, and pickup where the previous transfer left off for the specified unit address, based on the value of the continue-token.

- If continue-token is zero and if previous H_SEND_LOGICAL_LAN for the specified unit address was suspended with H_Busy and never completed, then cleanup the state from the previously suspended call before proceeding.
- Verifies the VLAN number -- else H_Parameter.
- Proceeds down the 6 buffer descriptors until the first one that has a length of 0
  - If the **"ibm,max-virtual-dma-size"** property exist in the **/vdevice** node of the device tree, then if the length is greater than the value of this property, return H_Parameter
  - For the length of the buffer:
    - Verifies the I/O buffer addresses translate through the sender's RTCE table else H_Parameter.
- Verifies the destination MAC address for the VLAN
  - If MAC address is not cached and there exists a Trunk Adapter for the VLAN, then flags the message as destined for the Trunk Adapter and continues processing
  - If MAC address is not cached and a Trunk Adapter does not exist for the VLAN, then drop the message (H_Dropped)
- For each Destination MAC Address (broadcast MAC address turns into multi-cast to all destinations on the specified VLAN):
  - In the case of multicast MAC addresses the following algorithm defines the members of the receiver class for a given VLAN:
    For each logical lan IOA that would be a target for a broadcast from the source IOA:
    - If the receiving IOA is not enabled for non-broadcast multicast frames then continue
    - If the receiving IOA is not enabled for filtering non-broadcast multicast frames then copy the frame to the IOA's receive buffer
    - Else
    -- If (lookup filter (table index)) then copy the frame to the IOA's receive buffer
    -- Else if the receiving IOA is not enabled for filtering non-broadcast multicast frames then copy the frame to the IOA's receive buffer /*allows for races on filter insertion */

    int lookup filter (table index)
    - Firmware implementation designed algorithm
  - Searches the receiver's receive queue for a suitable buffer and atomically dequeues it:
    - If no suitable buffer is found, the receiver's dropped packet counter (last 8 bytes of buffer list) is incremented and processing proceeds to the next receiver if any.
  - Copy the send data in to the selected receive buffer, build a receive queue entry, and generate an interrupt to the receiver if the interrupt is enabled.
- If any frames were dropped return H_Dropped else return H_Success.

Firmware Implementation Note: If during the processing of the H_SEND_LOGICAL_LAN call, it becomes necessary to temporarily suspend the processing of the call (for example, due to the length of time it is taking to process the call), the firmware may return a continuation token in R4, along with the return code of H_Busy. The value of the continuation token is up to the firmware, and will be passed back by the software as the continue-token parameter on the next call of H_SEND_LOGICAL_LAN.

This hcall() interlocks with H_MIGRATE_DMA to allow for migration of TCE mapped DMA pages.

Note: It is possible for either or both the sending and receiving OS to modify its RTCE tables so as to affect the TCE translations being actively used by H_SEND_LOGICAL_LAN. This is an error condition on the part of the OS. Implementations need only insure that such conditions do not corrupt memory in innocent partitions and should

not add path length to protect guilty partitions. By all means the path length of H_GET_TCE and H_PUT_TCE should not be increased. If reasonably possible, without significant path length addition, implementations should: On send buffer translation corruption, return H_Parameter to the sender and either totally drop the packet prior to reception, or if the receive buffer has been processed past the point of transparent recycling, mark the receive buffer as received in error in the receive queue. On receive buffer translation corruption, terminate the data copy to the receive buffer and mark the buffer as received in error in the receive queue.

16.4.3.6 H_MULTICAST_CTRL

This hcall() controls the reception of non-broadcast multicast packets (those with the high order address byte being odd but not the all 1's address). All implementations support the enabling and disabling of the reception of all multicast packets on their V-LAN. Additionally, the l-lan device driver through this call may ask the firmware to filter multicast packets for it. That is, receive packets only if they contain multicast addresses specified by the device driver. The number of simultaneous multicast packet filters supported is implementation dependent, and is specified in the **"ibm,mac-address-filters"** property of the l-lan device tree node. Therefore, the device driver must be prepared to have any filter request fail, and fall back to enabling reception of all multicast packets and filtering them in the device driver. Semantically, the device driver may ask that the reception of multicast packets be enabled or disabled, further if reception is enabled, they may be filtered by only allowing reception of packets who's mac address matches one of the entries in the filter table. The call also manages the contents of the mac address filter table. Individual mac addresses may be added, or removed, and the filter table may be cleared. If the filter table is modified by a call, there is the possibility that a packet may be improperly filtered (one that was to be filtered out may get through or one that should have gotten through may be dropped) this is done to avoid adding extra locking to the packet processing code. In most cases higher level protocols will handle the condition (since firmware filtering is simply a performance optimization), if, however, a specific condition requires complete accuracy, the device driver can disable filtering prior to an update, do its own filtering (as would be required if the number of receivers exceeded the number of filters in the filter table) update the filter table, and then reenable filtering.

Syntax:

```
int64                       /* H_Success,      Expected Return Code
                            /* H_Parameter,    One or more parameters were invalid */
                            /*H_Constrained,   The operation failed because of resource constraints (too many */
                            /*                 filter requests*/
                            /*H_Not_Found,     The requested remove object was not found*/
                            /*H_Hardware       The operation failed because of a hardware error*/
    hcall(const uint64 H_MULTICAST_CTRL,/* Controls multicast address filtering.*/
          uint64 unit-address,      /*As specified in the Logical LAN device tree node */
          uint64 flags,     /* Two sets of encoded flag fields control the function */
                            /* -- Bits 44-47 control the multicast enables */
                                  /* Bit 44 = 0 do not modify enable reception of multicast frames */
                                        /* value of bit 46 ignored */
                                  /* Bit 44 = 1 modify enable reception of multicast frames */
                                        /* if bit 46 = 1 allow reception of multicast frames*/
                                        /* if bit 46 = 0 prohibit reception of all multicast frames */
                                  /* Bit 45 = 0 do not modify filtering of multicast frames */
                                        /* value of bit 47 ignored */
                                  /* Bit 45 = 1 modify enable of filtering of multicast frames */
                                        /* if bit 47 = 1 if reception of multicast frames are allowed, */
                                        /*          further filter frames via mac address filter table */
                                        /* if bit 47 = 0 if reception of multicast frames are allowed, */
                                        /*          do not filter out any due to mac address filter table */
                            /* -- Bits 62 and 63 control modification of the mac address filter table */
                                  /*-- Bits 62 and 63 sub decode: */
                                        /* 00 Leave filter table unmodified */
```

```
                                        /* 01 Add specified MAC filter address to the table */
                                        /* 10 Remove specified MAC filter address from the table*/
                                        /* 11 Clear the MAC filter address table */
                                    /* All other flag bits are reserved */
        uint64 multicast-address)   /* 8 byte MAC multicast address to be enabled (11) or disabled (10) */
```

Parameters:

- unit-address: Unit Address per device tree node **"reg"** property
- flags: Only bits 44-47 and 62-63 are defined all other bits should be zero.
- multi-cast-address: Multicast MAC address, if flag bits 62 and 63 are 01 or 10, else this parameter is ignored.

Return value in register R4:

State of Enables and Count of MAC Filters in table.

Format:

Bits: 0----------45,46-47,48----------63

Reserved R F MAC Filter Count

R = The value of the Receipt Enable bit

F = The value of the Filter Enable bit

MAC Filter Count -- 16 bit count of the number of MAC Filters in the multicast filter table.

Semantics:

- Validate the unit-address parameter else return H_Parameter.
- Validate that no reserved flag bit = 1 else return H_Parameter.
- If any bits are on in the high order two bytes of the MAC parameter Return H_Parameter
- Modify Enables per specification if requested.
- Modify the Filter Table per specification if requested filtering is disable during any filter table modification and filter enable state restored after filter table modification).
  - If don't modify RC=H_Success
  - If Clear all: initialize the filter table, RC=H_Success
  - If Add:
    - If there is room in the table insert new MAC Filter entry, MAC Filter count++, RC=H_Success
    - Else RC=H_Constrained
    - (duplicates are silently dropped -- filter count stays the same RC=H_Success)
  - If Remove:
    - Locate the specified entry in the MAC Filter Table
    - If Found remove the entry, MAC Filter count--, RC=H_Success
    - Else RC=H_Not_Found

- Load the Enable Bits into R4 bits 46 and 47 Load the MAC Filter count into R4 Bits 48-63
- Return RC

16.4.3.7 H_CHANGE_LOGICAL_LAN_MAC

This hcall() allows the changing of the virtual IOA's MAC address.

Syntax:

```
int64                                   /* H_Success, H_Parameter, H_Hardware */
        hcall(const uint64 H_CHANGE_LOGICAL_LAN_MAC,/* Change the MAC address */
        uint64    unit-address,         /* As specified in the Logical LAN device tree node */
        uint64    mac-address)          /* New MAC address for the virtual IOA). */
```

Parameters:

- unit-address: Unit Address per device tree node **"reg"** property
- mac-address: The new receive filter MAC address

Semantics:

- Validates the unit address, else H_Parameter
- Records the low order 6 bytes of mac-address for filtering future incoming messages
- Returns H_Success

16.4.3.8 H_ILLAN_ATTRIBUTES

There are certain ILLAN attributes that are made visible to and can be manipulated by partition software. The H_ILLAN_ATTRIBUTES hcall is used to read and modify the attributes (see Section 16.4.3.8, "H_ILLAN_ATTRIBUTES," on page 564). Table 207, "ILLAN Attributes," on page 564 defines the attributes that are visible and manipulatable.

Table 207. ILLAN Attributes

Bit(s)	Field Name	Definition
0-49	Reserved	
50	Checksum Offload Padded Packet Support	This bit is implemented when the ILLAN Checksum Offload Padded Packet Support option is implemented. See Section 16.4.6.2.3, "Checksum Offload Padded Packet Support Option," on page 574. 0: Software must not request checksum offload, by setting Bit 6 of the buffer descriptor (the No Checksum bit), for packets that have been padded. 1: Software may request checksum offload, by setting Bit 6 of the buffer descriptor (the No Checksum bit), for packets that have been padded.

Table 207. ILLAN Attributes ***(Continued)***

Bit(s)	Field Name	Definition
51	Buffer Size Control	This bit is implemented when the ILLAN Buffer Size Control option is implemented. This bit allows the partition software to inhibit the use of too large of a buffer for incoming packets, when a reasonable size buffer is not available. The state of this bit cannot be changed between the time that the ILLAN is registered by an H_REGISTER_LOGICAL_LAN and it is deregistered by an H_FREE_LOGICAL_LAN. See also Section 16.4.6.3, “ILLAN Buffer Size Control Option,” on page 574. 1: The hypervisor will keep a history of what buffer sizes have been registered. When a packets arrives the history is searched to find the smallest buffers size that will contain the packet. If that buffer size is depleted then the packet is dropped by the hypervisor (H_Dropped) instead of searching for the next larger available buffer. 0: This is the initial value. When a packet arrives, the available buffers are searched for the smallest available buffer that will hold the packet, and the packet is not dropped unless no buffer is available in which the packet will fit.
52-55	Trunk Adapter Priority	This field is implemented for a VIOA whenever the ILLAN Backup Trunk Adapter option is implemented and the VIOA is a Trunk Adapter (the Active Trunk Adapter bit will be implemented, also, in this case). If this field is a 0, then the either the ILLAN Backup Trunk Adapter option is not implemented or it is implemented but this VIOA is not a Trunk Adapter. A non-0 value in this field reflects the priority of the node in the backup Trunk Adapter hierarchy, with a value of 1 being the highest (most favored) priority, the value of 2 being the next highest priority, and so on. This field may or may not be changeable by the partition firmware via the H_ILLAN_ATTRIBUTES hcall() (platform implementation dependent). If not changeable, then attempts to change this field will result in a return code of H_Constrained. See also Section 16.4.6.1, “ILLAN Backup Trunk Adapter Option,” on page 571.
56-60	Reserved	
61	TCP Checksum Offload Support for IPv6	This bit is implemented for a VIOA whenever the ILLAN Checksum Offload Support option is implemented for TCP, the IPv6 protocol, and the following extension headers: ◆ Hop-by-Hop Options ◆ Routing ◆ Destination Options ◆ Authentication ◆ Mobility This bit is initially set to 0 by the firmware and the ILLAN DD may attempt to set it to a 1 by use of the H_ILLAN_ATTRIBUTES hcall() if the DD supports the option for TCP and IPv6. Firmware will not allow changing the state of this bit if it does not support Checksum Offload Support for TCP for IPv6 for the VIOA (H_Constrained would be returned in this case from the H_ILLAN_ATTRIBUTES hcall() when this bit is a 1 in the set-mask). This state of this bit cannot be changed between the time that the ILLAN is registered by an H_REGISTER_LOGICAL_LAN and it is deregistered by an H_FREE_LOGICAL_LAN. See Section 16.4.6.2, “ILLAN Checksum Offload Support Option,” on page 572 for more information. 1: The partition software has indicated that it supports the ILLAN Checksum Offload Support option for TCP and IPv6 protocol and for the above stated extension headers by using the H_ILLAN_ATTRIBUTES hcall() with this bit set to a 1 in the set-mask, and the firmware has verified that it supports this protocol for the option for the VIOA. 0: The partition software has not indicated that it supports the ILLAN Checksum Offload Support option for TCP and IPv6 protocol and for the above stated extension headers by using the H_ILLAN_ATTRIBUTES hcall() with this bit set to a 1 in the set-mask, or it has but the firmware does not support the option, or supports the option but not for this protocol or for this VIOA.

Table 207. ILLAN Attributes *(Continued)*

Bit(s)	Field Name	Definition
62	TCP Checksum Offload Support for IPv4	This bit is implemented for a VIOA whenever the ILLAN Checksum Offload Support option is implemented for TCP and the IPv4 protocol. This bit is initially set to 0 by the firmware and the ILLAN DD may attempt to set it to a 1 by use of the H_ILLAN_ATTRIBUTES hcall() if the DD supports the option for TCP and IPv4. Firmware will not allow changing the state of this bit if it does not support Checksum Offload Support for TCP or IPv4 for the VIOA (H_Constrained would be returned in this case from the H_ILLAN_ATTRIBUTES hcall() when this bit is a 1 in the set-mask). This state of this bit cannot be changed between the time that the ILLAN is registered by an H_REGISTER_LOGICAL_LAN and it is deregistered by an H_FREE_LOGICAL_LAN. See Section 16.4.6.2, “ILLAN Checksum Offload Support Option,” on page 572 for more information. 1: The partition software has indicated that it supports the ILLAN Checksum Offload Support option for TCP and IPv4 protocol by using the H_ILLAN_ATTRIBUTES hcall() with this bit set to a 1 in the set-mask, and the firmware has verified that it supports this protocol for the option for the VIOA. 0: The partition software has not indicated that it supports the ILLAN Checksum Offload Support option for TCP and IPv4 by using the H_ILLAN_ATTRIBUTES hcall() with this bit set to a 1 in the set-mask, or it has but the firmware does not support the option, or supports the option but not for this protocol or for this VIOA.
63	Active Trunk Adapter	This bit is implemented for a VIOA whenever the ILLAN Backup Trunk Adapter option is implemented and the VIOA is a Trunk Adapter (the Trunk Adapter Priority field will be implemented, also, in this case). ◆ This bit is initially set to 0 by the firmware for an inactive Trunk Adapter. ◆ This bit is initially set to 1 by the firmware for an active Trunk Adapter. ◆ This bit will be changed from a 0 to a 1 when all the following a true: (1) the partition software (via the H_ILLAN_ATTRIBUTES hcall() with this bit set to a 1 in the set-mask) attempts to set this bit to a 1, (2) the firmware supports the Backup Trunk Adapter option, (3) the VIOA is a Trunk Adapter. ◆ This bit will be changed from a 1 to a 0 by the firmware when another Trunk Adapter has had its Active Trunk Adapter bit changed from a 0 to a 1. See Section 16.4.6.1, “ILLAN Backup Trunk Adapter Option,” on page 571 for more information. 1: The VIOA is the active Trunk Adapter. 0: The VIOA is not an active Trunk Adapter or is not a Trunk Adapter at all.

R1–16.4.3.8–1. If the H_ILLAN_ATTRIBUTES hcall is implemented, then it must implement the attributes as they are defined in Table 207, “ILLAN Attributes,” on page 564 and the syntax and semantics as defined in Section 16.4.3.8, “H_ILLAN_ATTRIBUTES,” on page 564.

R1–16.4.3.8–2. The H_ILLAN_ATTRIBUTES hcall must ignore bits in the set-mask and reset-mask which are not implemented for the specified unit-address and must process as an exception those which cannot be changed for the specified unit-address (H_Constrained returned), and must return the following for the ILLAN Attributes in R4:

a. A value of 0 for unimplemented bit positions.

b. The resultant field values for implemented fields.

Syntax:

```
uint64      /*  H_Success       Expected return code */
            /*  H_Parameter     One or more parameters were in error */
            /*  H_Constrained   One or more parameters was not changeable to the value requested*/
            /*                  the result was constrained to a legitimate value for the implementation*/
            /*  H_Hardware      Operation failed because of hardware error*/
  hcall (   const uint64 H_ILLAN_ATTRIBUTES,/* Returns in R4 the resulting ILLAN Attributes*/
```

```
uint64   unit-address,    /* The ILLAN Unit Address on which to perform this operation*/
uint64   reset-mask,      /* Mask of Attribute bits to be reset*/
uint64   set-mask);       /* Mask of Attribute bits to be set*/
```

Parameters:

unit-address: Unit Address per device tree node **"reg"** property. The ILLAN unit address on which this Attribute modification is to be performed.

reset-mask: The bit-significant mask of bits to be reset in the ILLAN's Attributes (the reset-mask bit definition aligns with the bit definition of the ILLAN's Attributes, as defined in Table 207, "ILLAN Attributes," on page 564). The complement of the reset-mask is ANDed with the ILLAN's Attributes, prior to applying the set-mask. See semantics for more details on any field-specific actions needed during the reset operations. If a particular field position in the ILLAN Attributes is not implemented, then the corresponding bit(s) in the reset-mask are ignored.

set-mask: The bit-significant mask of bits to be set in the ILLAN's Attributes (the set-mask bit definition aligns with the bit definition of the ILLAN's Attributes, as defined in Table 207, "ILLAN Attributes," on page 564). The set-mask is ORed with the ILLAN's Attributes, after to applying the reset-mask. See semantics for more details on any field-specific actions needed during the set operations. If a particular field position in the ILLAN Attributes is not implemented, then the corresponding bit(s) in the set-mask are ignored.

Semantics:

- Validate that Unit Address belongs to the partition, else H_Parameter.
- Reset/set the bits in the ILLAN Attributes, as indicated by the rest-mask and set-mask except as indicated in the following conditions.
- If the Buffer Size Control bit is trying to be changed from a 0 to a 1 and any of the following is true, then do not allow the change (H_Constrained will be returned):
  - The ILLAN is active. That is, the ILLAN has been registered (H_REGISTER_LOGICAL_LAN) but has not be deregistered (H_FREE_LOGICAL_LAN).
  - The firmware does not support the ILLAN Buffer Size Control option.
- If the Buffer Size Control bit is trying to be changed from a 1 to a 0 and any of the following is true, then do not allow the change (H_Constrained will be returned):
  - The ILLAN is active. That is, the ILLAN has been registered (H_REGISTER_LOGICAL_LAN) but has not be deregistered (H_FREE_LOGICAL_LAN).
- If either the TCP Checksum Offload Support for IPv4 bit or TCP Checksum Offload Support for IPv6 bit is trying to be changed from a 0 to a 1 and any of the following is true, then do not allow the change (H_Constrained will be returned):
  - The ILLAN is active. That is, the ILLAN has been registered (H_REGISTER_LOGICAL_LAN) but has not be deregistered (H_FREE_LOGICAL_LAN).
  - The firmware does not support the ILLAN Checksum Offload Support option or supports it, but not for the specified protocol(s) or does not support it for this VIOA.
- If the TCP Checksum Offload Support for IPv4 bit or TCP Checksum Offload Support for IPv6 bit is trying to be changed from a 1 to a 0 and any of the following is true, then do not allow the change (H_Constrained will be returned):
  - The ILLAN is active. That is, the ILLAN has been registered (H_REGISTER_LOGICAL_LAN) but has not be deregistered (H_FREE_LOGICAL_LAN).

- If the Active Trunk Adapter bit is trying to be changed from a 0 to a 1 and any of the following is true, then do not allow the change (H_Constrained will be returned):
  - The firmware does not support the ILLAN Backup Trunk Adapter option or this VIOA is not a Trunk Adapter.
- If the Active Trunk Adapter bit is trying to be changed from a 1 to a 0, then return H_Parameter.
- If the Active Trunk Adapter bit is changed from a 0 to a 1 for a VIOA, then also set any previously active Trunk Adapter's Active Trunk Adapter bit from a 1 to a 0.
- If the Trunk Adapter Priority field is trying to be changed from 0 to a non-0 value, then return H_Parameter.
- If the Trunk Adapter Priority field is trying to be changed from a non-0 value to another non-0 value and either the parameter is not changeable or the change is not within the platform allowed limits, then do not allow the change (H_Constrained will be returned).
- Load R4 with the value of the ILLAN's Attributes, with any unimplemented bits set to 0, and if all requested changes were made then return H_Success, otherwise return H_Constrained.

16.4.3.9 Other hcall()s extended or used by the Logical LAN Option

16.4.3.9.1 H_VIO_SIGNAL

The H_VIO_SIGNAL hcall() is used by multiple VIO options.

16.4.3.9.2 H_EOI

The H_EOI hcall(), when specifying an interrupt source number associated with an interpartion logical LAN IOA, incorporates the interrupt reset function.

16.4.3.9.3 H_XIRR

This call is extended to report the virtual interrupt source number associated with virtual interrupts associated with an ILLAN IOA.

16.4.3.9.4 H_PUT_TCE

This standard hcall() is used to manage the ILLAN IOA's I/O translations.

16.4.3.9.5 H_GET_TCE

This standard hcall() is used to manage the ILLAN IOA's I/O translations.

16.4.3.9.6 H_MIGRATE_DMA

This hcall() is extended to serialize with the H_SEND_LOGICAL_LAN hcall() to allow for migration of TCE mapped DMA pages.

16.4.4 RTAS Calls Extended or Used by the Logical LAN Option

Platforms may combine the Logical LAN option with most other LoPAPR options such as dynamic reconfiguration by including the appropriate OF properties and extending the associated firmware calls. However, the *ibm,set-xive, ibm,get-xive, ibm,int-off,* and *ibm,int-on* RTAS calls are extended as part of the base support.

16.4.5 Interpartition Logical LAN Requirements

The following requirements are mandated for platforms implementing the ILLAN option.

R1–16.4.5–1. For the ILLAN option: The platform must interpret logical LAN buffer descriptors as defined in Section 16.4.1.1, “Buffer Descriptor,” on page 553.

R1–16.4.5–2. For the ILLAN option: The platform must reject logical LAN buffer descriptors that are not 8 byte aligned.

R1–16.4.5–3. For the ILLAN option: The platform must interpret the first byte of a logical LAN buffer descriptor as a control byte, the high order bit being the valid bit.

R1–16.4.5–4. For the ILLAN option: The platform must set the next to high order bit of the control byte of the logical LAN buffer descriptor for the receive queue to the inverse of the value currently being used to indicate a valid receive queue entry.

R1–16.4.5–5. For the ILLAN option: The platform must interpret the 2nd through 4th bytes of a logical LAN buffer descriptor as the binary length of the buffer in I/O space (relative to the TCE mapping table defined by the logical IOA’s **`“ibm,my-dma-window”`** property).

R1–16.4.5–6. For the ILLAN option: The platform must interpret the 5th through 8th bytes of a logical LAN buffer descriptor as the binary beginning address of the buffer in I/O space (relative to the TCE mapping table defined by the logical IOA’s **`“ibm,my-dma-window”`** property).

R1–16.4.5–7. For the ILLAN option: The platform must interpret logical LAN Buffer Lists as defined in Section 16.4.1.2, “Buffer List,” on page 553.

R1–16.4.5–8. For the ILLAN option: The platform must reject logical LAN Buffer Lists that are not mapped relative to the TCE mapping table defined by the logical IOA’s **`“ibm,my-dma-window”`** property.

R1–16.4.5–9. For the ILLAN option: The platform must reject logical LAN buffer lists that are not 4 KB aligned.

R1–16.4.5–10. For the ILLAN option: The platform must interpret the first 8 bytes of a logical LAN buffer list as a buffer descriptor for the logical IOA’s Receive Queue.

R1–16.4.5–11. For the ILLAN option: The platform must interpret the logical LAN receive queue as defined in Section 16.4.1.3, “Receive Queue,” on page 553.

R1–16.4.5–12. For the ILLAN option: The platform must reject a logical LAN receive queue that is not mapped relative to the TCE mapping table defined by the logical IOA’s **`“ibm,my-dma-window”`** property.

R1–16.4.5–13. For the ILLAN option: The platform must reject a logical LAN receive queue that is not aligned on a 4 byte boundary.

R1–16.4.5–14. For the ILLAN option: The platform must reject a logical LAN receive queue that is not an exact multiple of 12 bytes long.

R1–16.4.5–15. For the ILLAN option: The platform must manage the logical LAN receive queue as a circular buffer.

R1–16.4.5–16. For the ILLAN option: The platform must enqueue 12 byte logical LAN receive queue entries when a new message is received.

R1–16.4.5–17. For the ILLAN option: The platform must set the last 8 bytes of the logical LAN receive queue entry to the value of the user supplied correlator found in the first 8 bytes of the logical LAN receive buffer used to contain the message before setting the first 4 bytes of the logical LAN receive queue entry.

R1–16.4.5–18. For the ILLAN option: The platform must set the first 4 bytes of the logical LAN receive queue entry such that the first byte contains the control field (high order bit the inverse of the valid toggle in the receive queue buffer descriptor, next bit to a one if the message payload is valid) and the last 3 bytes contains the receive message length, after setting the correlator field in the last 8 bytes per Requirement R1–16.4.5–17.

R1–16.4.5–19. For the ILLAN option: The platform must when crossing from the end of the logical LAN receive queue back to the beginning invert the value of the valid toggle in the receive queue buffer descriptor.

R1–16.4.5–20. For the ILLAN option: The platform's OF must disable interrupts from the logical LAN IOA before initially passing control to the booted client program.

R1–16.4.5–21. For the ILLAN option: The platform must present (as appropriate per RTAS control of the interrupt source number) the partition owning a logical LAN receive queue the appearance of an interrupt, from the interrupt source number associated, through the OF device tree node, with the virtual device, when a new entry is enqueued to the logical LAN receive queue and the last interrupt mode set via the H_VIO_SIGNAL was "Enabled", unless a previous interrupt from the interrupt source number is still outstanding.

R1–16.4.5–22. For the ILLAN option: The platform must NOT present the partition owning a logical LAN receive queue the appearance of an interrupt, from the interrupt source number associated, through the OF device tree node, with the virtual device, if the last interrupt mode set via the H_VIO_SIGNAL was "Disabled", unless a previous interrupt from the interrupt source number is still outstanding.

R1–16.4.5–23. For the ILLAN option: The platform must interpret logical LAN receive buffers as defined in Section 16.4.1.5, "Receive Buffers," on page 555.

R1–16.4.5–24. For the ILLAN option: The platform must reject a logical LAN receive buffer that is not mapped relative to the TCE mapping table defined by the logical IOA's **`"ibm,my-dma-window"`** property.

R1–16.4.5–25. For the ILLAN option: The platform must reject a logical LAN receive buffer that is not aligned on a 4 byte boundary.

R1–16.4.5–26. For the ILLAN option: The platform must reject a logical LAN receive buffer that is not a minimum of 16 bytes long.

R1–16.4.5–27. For the ILLAN option: The platform must not modify the first 8 bytes of a logical LAN receive buffer, this area is reserved for a user supplied correlator value.

R1–16.4.5–28. For the ILLAN option: The platform must not allow corruption caused by a user modifying the logical LAN receive buffer from escaping the user partition (except as a side effect of some another user partition I/O operation).

R1–16.4.5–29. For the ILLAN option: The platform's **`l-lan`** OF device tree node must contain properties as defined in Table 206, "Properties of the Logical LAN OF Device Tree Node," on page 556. (Other standard I/O adapter properties are permissible as appropriate.)

R1–16.4.5–30. For the ILLAN option: The platform must implement the H_REGISTER_LOGICAL_LAN hcall() as defined in Section 16.4.3.1, "H_REGISTER_LOGICAL_LAN," on page 557.

R1–16.4.5–31. For the ILLAN option: The platform must implement the H_FREE_LOGICAL_LAN hcall() as defined in Section 16.4.3.2, "H_FREE_LOGICAL_LAN," on page 558.

R1–16.4.5–32. For the ILLAN option: The platform must implement the H_ADD_LOGICAL_LAN_BUFFER hcall() as defined in Section 16.4.3.3, "H_ADD_LOGICAL_LAN_BUFFER," on page 559.

R1–16.4.5–33. For the ILLAN option: The platform must implement the H_SEND_LOGICAL_LAN hcall() as defined in Section 16.4.3.5, "H_SEND_LOGICAL_LAN," on page 560.

R1–16.4.5–34. For the ILLAN option: The platform must implement the H_SEND_LOGICAL_LAN hcall() such that an OS requested modification to an active RTCE table entry cannot corrupt memory in other partitions. (Except indirectly as a result of some other of the partition's I/O operations.)

R1–16.4.5–35. For the ILLAN option: The platform must implement the H_CHANGE_LOGICAL_LAN_MAC hcall() as defined in Section 16.4.3.7, "H_CHANGE_LOGICAL_LAN_MAC," on page 564.

R1–16.4.5–36. For the ILLAN option: The platform must implement the H_VIO_SIGNAL hcall() as defined in Section 17.2.1.3, "VIO Interrupt Control," on page 602.

R1–16.4.5–37. For the ILLAN option: The platform must implement the extensions to the H_EOI hcall() as defined in Section 16.4.3.9.2, "H_EOI," on page 568.

R1–16.4.5–38. For the ILLAN option: The platform must implement the extensions to the H_XIRR hcall() as defined in Section 16.4.3.9.3, "H_XIRR," on page 568.

R1–16.4.5–39. For the ILLAN option: The platform must implement the H_PUT_TCE hcall().

R1–16.4.5–40. For the ILLAN option: The platform must implement the H_GET_TCE hcall().

R1–16.4.5–41. For the ILLAN option: The platform must implement the extensions to the H_MIGRATE_DMA hcall() as defined in Section 16.4.3.9.6, "H_MIGRATE_DMA," on page 568.

R1–16.4.5–42. For the ILLAN option: The platforms must emulate the standard PowerPC External Interrupt Architecture for the interrupt source numbers associated with the virtual devices via the standard RTAS and hypervisor interrupt calls.

16.4.6 Logical LAN Options

The ILLAN option has several sub-options. The hypervisor reports to the partition software when it supports one or more of these options, and potentially other information about those option implementations, via the implementation of the appropriate bits in the ILLAN Attributes, which can be ascertained by the H_ILLAN_ATTRIBUTES hcall(). The same hcall() may be used by the partition software to communicate back to the firmware the level of support for those options where the firmware needs to know the level of partition software support. The **`"ibm,illan-options"`** property will exist in the VIOA's Device Tree node, indicating that the H_ILLAN_ATTRIBUTES hcall() is implemented, and therefore that one or more of the options are implemented. The following sections give more details.

16.4.6.1 ILLAN Backup Trunk Adapter Option

The ILLAN Backup Trunk Adapter option allows the platform to provide one or more backups to a Trunk Adapter, for reliability purposes. Implementation of the ILLAN Backup Trunk Adapter option is specified to the partition by the existence of the **`"ibm,illan-options"`** property in the VIOA's Device Tree node and a non-0 value in the ILLAN Attributes Backup Trunk adapter Priority field. A Trunk Adapter becomes the active Trunk Adapter by calling H_ILLAN_ATTRIBUTES hcall() and setting its Active Trunk Adapter bit. Only one Trunk Adapter is active for a VLAN at a time. The protocols which determine which Trunk Adapter is active at any particular time, is beyond the scope of this architecture.

R1–16.4.6.1–1. For the ILLAN Backup Trunk Adapter option: The platform must implement the ILLAN option.

R1–16.4.6.1–2. For the ILLAN Backup Trunk Adapter option: The platform must implement the H_ILLAN_ATTRIBUTES hcall().

R1–16.4.6.1–3. For the ILLAN Backup Trunk Adapter option: The platform must implement the **`"ibm,illan-options"`** and **`"ibm,trunk-adapter"`** properties in all the Trunk Adapter nodes of the Device Tree.

R1–16.4.6.1–4. For the ILLAN Backup Trunk Adapter option: The platform must implement the Active Trunk Adapter bit and the Backup Trunk Adapter Priority field in the ILLAN Attributes, as defined in Table 207, "ILLAN Attributes," on page 564, for all Trunk Adapter VIOAs.

R1–16.4.6.1–5. For the ILLAN Backup Trunk Adapter option: The platform must allow only one Trunk Adapter to be active for a VLAN at any given time, and must:

a. Make the determination of which one is active by whichever was the most recent one to set its Active Trunk Adapter bit in their ILLAN Attributes.

b. Turn off the Active Trunk Adapter bit in the ILLAN Attributes for a Trunk Adapter when it is removed from the active Trunk Adapter state.

16.4.6.2 ILLAN Checksum Offload Support Option

This option allows for the support of IOAs that do checksum offload processing. This option allows for support at one end (client or server) but not the other, on a per-protocol basis, with the hypervisor generating the checksum when the client supports offload but the server does not, and the operation is a send from the client.

16.4.6.2.1 General

The H_ILLAN_ATTRIUBTES hcall is used to establish the common set of checksum offload protocols to be supported between the firmware and the partition software. The firmware indicates support for H_ILLAN_ATTRIBUTES via the **"ibm,illan-options"** property in the VIOA's Device Tree node. The partition software can determine which of the Checksum Offload protocols (if any) that the firmware supports by either attempting to set the bits in the ILLAN Attributes of the protocols that the partition software supports or by calling the hcall() with reset-mask and set-mask parameters of all-0's (the latter being just a query and not a request to support anything between the partition and the firmware).

Two bits in the control field of the first buffer descriptor specify which operations do not contain a checksum and which have had their checksum already verified. See Section 16.4.1.1, "Buffer Descriptor," on page 553. These two bits get transferred to the corresponding control field of the Receive Queue Entry, with the exception that the H_SEND_LOGICAL_LAN hcall will sometimes set these to 0b00 (see Section 16.4.6.2.2, "H_SEND_LOGICAL_LAN Semantic Changes," on page 572).

R1–16.4.6.2.1–1. For the ILLAN Checksum Offload Support option: The platform must do all the following:

a. Implement the ILLAN option.

b. Implement the H_ILLAN_ATTRIBUTES hcall().

c. Implement the **"ibm,illan-options"** property in the VIOA's Device Tree node.

d. Implement the appropriate Checksum Offload Support bit(s) of the ILLAN Attributes, as defined in Table 207, "ILLAN Attributes," on page 564.

Software Implementation Note: Fragmentation and encryption are not supported when the No Checksum bit of the Buffer Descriptor is set to a 1.

16.4.6.2.2 H_SEND_LOGICAL_LAN Semantic Changes

There are several H_SEND_LOGICAL_LAN semantic changes required for the ILLAN Checksum Offload Support option. See Section 16.4.3.5, "H_SEND_LOGICAL_LAN," on page 560 for the base semantics.

R1–16.4.6.2.2–1. For the ILLAN Checksum Offload Support option: The H_SEND_LOGICAL_LAN semantics must be changed as follows:

a. As shown in Table 208, “Summary of H_SEND_LOGICAL_LAN Semantics with Checksum Offload,” on page 573, and for multi-cast operations, the determination in this table must be applied for each destination.

b. If the No Checksum bit is set to a 1 in the first buffer descriptor and the adapter is not a Trunk Adapter, and the source MAC address does not match the adapter's MAC address, then drop the packet.

Table 208. Summary of H_SEND_LOGICAL_LAN Semantics with Checksum Offload

Has Sender Set the Appropriate Checksum Offload Support bit in the ILLAN Attributes for the Protocol Being Used?	Has Receiver Set the Appropriate Checksum Offload Support bit in the ILLAN Attributes for the Protocol Being Used?	No Checksum bit in the Buffer Descriptor	Checksum Good bit in the Buffer Descriptor	H_SEND_LOGICAL_LAN Additional Semantics	Receiver DD Additional Requirements
no	-	0	0	None.	None.
no	-	Either bit non-0		Return H_Parameter	
yes	-	0	0	None.	None.
yes	no	0	1	Set the No Checksum and Checksum Good bits in the Buffer Descriptor to 00 on transfer.	None.
yes	no	1	1	Generate checksum and set the No Checksum and Checksum Good bits in the Buffer Descriptor to 00 on transfer.	None.
yes	yes	0	1	None.	Do not need to do checksum checking.
yes	yes	1	1	None.	Do not need to do checksum checking. Generate checksum if packet is to be passed on to an external LAN (may be done by the IOA or by the DD).
-	-	1	0	Return H_Parameter	
yes	-	01 or 11 and packet type not supported by the hypervisor, as indicated by the value returned by the H_ILLAN_ATTRIBUTES hcall()		Return H_Parameter	

R1–16.4.6.2.2–2. For the ILLAN Checksum Offload Support option: The Receiver DD Additional Requirements shown in Table 208, “Summary of H_SEND_LOGICAL_LAN Semantics with Checksum Offload,” on page 573 must be implemented.

R1–16.4.6.2.2–3. For the ILLAN Checksum Offload Support option: When the caller of H_SEND_LOGICAL_LAN has set the No Checksum bit in the Control field to a 1, then they must also have set the checksum field in the packet to 0.

16.4.6.2.3 Checksum Offload Padded Packet Support Option

Firmware may or may not support checksum offload for IPv4 packets that have been padded. The Checksum Offload Padded Packet Support bit of the ILLAN Attributes specifies whether or not this option is supported.

R1–16.4.6.2.3–1. For the Checksum Offload Padded Packet Support Option: The platform must do all the following:

a. Implement the ILLAN Checksum Offload Support option.

b. Implement the Checksum Offload Padded Support bit of the ILLAN Attributes, as defined in Table 207, "ILLAN Attributes," on page 564, and set that bit to a value of 1.

16.4.6.3 ILLAN Buffer Size Control Option

It is the partition software's responsibility to keep firmware supplied with enough buffers to keep packets from being dropped. The ILLAN Buffer Size Control option gives the partition software a way to prevent a flood of small packets from consuming buffers that have been allocated for larger packets.

When this option is implemented and the Buffer Size Control bit in the ILLAN Attributes is set to a 1 for the VLAN, the hypervisor will keep a history of what buffer sizes have been registered. Then, when a packets arrives the history is searched to find the smallest buffer size that will contain the packet. If that buffer size is depleted then the packet is dropped by the hypervisor (H_Dropped) instead of searching for the next larger available buffer.

16.4.6.3.1 General

The following are the general requirements for this option. For H_SEND_LOGICAL_LAN changes, see Section 16.4.6.2.2, "H_SEND_LOGICAL_LAN Semantic Changes," on page 572.

R1–16.4.6.3.1–1. For the ILLAN Buffer Size Control option: The platform must do all the following:

a. Implement the ILLAN option.

b. Implement the H_ILLAN_ATTRIBUTES hcall().

c. Implement the **`"ibm,illan-options"`** property in the VIOA's Device Tree node.

d. Implement the Buffer Size Control bit of the ILLAN Attributes, as defined in Table 207, "ILLAN Attributes," on page 564.

16.4.6.3.2 H_SEND_LOGICAL_LAN Semantic Changes

The following are the required semantic changes to the H_SEND_LOGICL_LAN hcall().

R1–16.4.6.3.2–1. For the ILLAN Buffer Size Control option: When the Buffer Size Control bit of the target of an H_SEND_LOGIC_LAN hcall() is set to a 1, then the firmware for the H_SEND_LOGICAL_LAN hcall() must not just search for any available buffer into which the packet will fit, but must instead only place the packet into the receiver's buffer if there is an available buffer of the smallest size previously registered by the receiver which will fit the packet, and must drop the packet for that target otherwise.

16.5 Virtual SCSI (VSCSI)

Virtual SCSI (VSCI) support is provided by code running in a server partition that uses the mechanisms of the Reliable Command/Response Transport and Logical Remote DMA of the Synchronous VIO Infrastructure to service I/O requests for code running in a client partition, such that, the client partition appears to enjoy the services of its own SCSI adapter (see Section 17.2.3, “Partition Managed Class - Synchronous Infrastructure,” on page 637). The terms server and client partitions refer to platform partitions that are respectively servers and clients of requests, usually I/O operations, using the physical I/O adapters (IOAs) that are assigned to the server partition. This allows a platform to have more client partitions than it may have physical I/O adapters because the client partitions share I/O adapters via the server partition.

The VSCSI architecture is built upon the architecture specified in the following sections:

- Section 17.2.1, “VIO Infrastructure - General,” on page 600
- Section 17.2.2, “Partition Managed Class Infrastructure - General,” on page 620
- Section 17.2.3, “Partition Managed Class - Synchronous Infrastructure,” on page 637

16.5.1 VSCSI General

This section contains an informative outline of the architectural intent of the use of the Synchronous VIO Infrastructure to provide VSCSI support, along with a few architectural requirements. Other implementations of the server and client partition code, consistent with this architecture, are possible and may be preferable.

The architectural metaphor for the VSCSI subsystem is that the server partition provides the virtual equivalent of a single SCSI DASD/Media string via each VSCSI server virtual IOA. The client partition provides the virtual equivalent of a single port SCSI adapter via each VSCSI client IOA. The platform, through the partition definition, provides means for defining the set of virtual IOA’s owned by each partition and their respective location codes. The platform also provides, through partition definition, instructions to connect each client partition’s VSCSI client IOA to a specific server partition’s VSCSI server IOA. That is, the equivalent of connecting the adapter cable to the specific DASD/Media string. The mechanism for specifying this partition definition is beyond the scope of this architecture. The human readable handle associated with the partition definition of virtual IOAs and their associated interconnection and resource configuration is the virtual location code. The OF unit address (**`Unit ID)`** remains the invariant handle upon which the OS builds its “physical to logical” configuration.

The client partition’s device tree contains one or more nodes notifying the partition that it has been assigned one or more virtual adapters. The node’s **`“type”`** and **`“compatible”`** properties notify the partition that the virtual adapter is a VSCSI adapter. The **`unit address`** of the node is used by the client partition to map the virtual device(s) to the OS’s corresponding logical representations. The **`“ibm,my-dma-window”`** property communicates the size of the RTCE table window panes that the hypervisor has allocated. The node also specifies the interrupt source number that has been assigned to the Reliable Command/Response Transport connection and the RTCE range that the client partition device driver may use to map its memory for access by the server partition via Logical Remote DMA. The client partition, uses the four hcall()s associated with the Reliable Command/Response Transport facility to register and deregister its CRQ, manage notification of responses, and send command requests to the server partition.

The server partition’s device tree contains one or more node(s) notifying the partition that it is requested to supply VSCSI services for one or more client partitions. The unit address (**`Unit ID)`** of the node is used by the server partition to map to the local logical devices that are represented by this VSCSI device. The node also specifies the interrupt source number that has been assigned to the Reliable Command/Response Transport connection and the RTCE range that the server partition device driver may use for its copy Logical Remote DMA. The server partition uses the four hcall()s associated with the Reliable Command/Response Transport facility to register and deregister its Command request queue, manage notification of new requests, and send responses back to the client partition. In addition, the

server partition uses the hcall()s of the Logical Remote DMA facility to manage the movement of commands and data associated with the client requests.

The client partition, upon noting the device tree entry for the virtual adapter, loads the device driver associated with the value of the **"compatible"** property. The device driver, when configured and opened, allocates memory for its CRQ (an array, large enough for all possible responses, of 16 byte elements), pins the queue and maps it into the I/O space of the RTCE window specified in the **"ibm,my-dma-window"** property using the standard kernel mapping services that subsequently use the H_PUT_TCE hcall(). The queue is then registered using the H_REG_CRQ hcall(). Next, I/O request control blocks (within which the I/O requests commands are built) are allocated, pinned, and mapped into I/O address space. Finally, the device driver registers to receive control when the interrupt source specified in the virtual IOA's device tree node signals.

Once the CRQ is setup, the device driver queues an Initialization Command/Response with the second byte of "Initialize" in order to attempt to tell the hosting side that everything is setup on the hosted side. The response to this send may be that the send has been dropped or has successfully been sent. If successful, the sender should expect back an Initialization Command/Response with a second byte of "Initialization Complete," at which time the communication path can be deemed to be open. If dropped, then the sender waits for the receipt of an Initialization Command/Response with a second byte of "Initialize," at which time an "Initialization Complete" message is sent, and if that message is sent successfully, then the communication path can be deemed to be open.

When the VSCSI Adapter device driver receives an I/O request from one of the SCSI device head drivers, it executes the following sequence. First an I/O request control block is allocated. Then it builds the SCSI request within the control block, adds a correlator field (to be returned in the subsequent response), I/O maps any target memory buffers and places their DMA descriptors into the I/O request control block. With the request constructed in the I/O request control block, the driver constructs a DMA descriptor (Starting Offset, and length) representing the I/O request within the I/O request control block. Lastly, the driver passes the I/O request's DMA descriptor to the server partition using the H_SEND_CRQ hcall(). Provided that the H_SEND_CRQ hcall() succeeds, the VSCSI Adapter device driver returns, waiting for the response interrupt indicating that a response has been posted by the server partition to the device driver's response queue. The response queue entry contains the summary status and request correlator. From the request correlator, the device driver accesses the I/O request control block, and from the summary status, the device driver determines how to complete the processing of the I/O request.

Notice that the client partition only uses the Reliable Command/Response Transport primitives; it does not use the Logical Remote DMA primitives. Since the server partition's RTCE tables are not authorized for access by the client partition, any attempt by the client partition to modify server partition memory would be prevented by the hypervisor. RTCE table access is granted on a connection by connection basis (client/server virtual device pair). If a client partition happens to be serving some other logical device, then the partition is entitled to use Logical Remote DMA for the virtual devices that is serving.

The server partition, upon noting the device tree entry for the virtual adapter, loads the device driver associated with the value of the **"compatible"** property. The device driver, when configured and opened, allocates memory for its request queue (an array, large enough for all possible outstanding requests, of 16 byte elements). The driver then pins the queue and maps it into I/O space, via the kernel's I/O mapping services that invoke the H_PUT_TCE hcall(), using the first window pane specified in the **"ibm,my-dma-window"** property. The queue is then registered using the H_REG_CRQ hcall(). Next, I/O request control blocks (within which the I/O request commands are built) are allocated, pinned, and I/O mapped. Finally the device driver registers to receive control when the interrupt source specified in the virtual IOA's device tree node signals.

Once the CRQ is setup, the device driver queues an Initialization Command/Response with the second byte of "Initialize" in order to attempt to tell the hosted side that everything is setup on the hosting side. The response to this send may be that the send has been dropped or has successfully been sent. If successful, the sender should expect back an Initialization Command/Response with a second byte of "Initialization Complete," at which time the communication path can be deemed to be open. If dropped, then the sender waits for the receipt of an Initialization Command/Response with a second byte of "Initialize," at which time an "Initialization Complete" message is sent, and if that message is sent successfully, then the communication path can be deemed to be open.

When the server partition's device driver receives an I/O request from its corresponding client partition's VSCSI adapter drivers, it is notified via the interrupt registered for above. The server partition's device driver selects an I/O request control block for the requested operation. It then uses the DMA descriptor from the request queue element to transfer the SCSI request from the client partition's I/O request control block to its own (allocated above), using the H_COPY_RDMA hcall() through the second window pane specified in the **"ibm,my-dma-window"** property. The server partition's device driver then uses kernel services, that are extended, to register the I/O request's DMA descriptors into extended capacity cross memory descriptors (ones capable of recording the DMA descriptors). These cross memory descriptors are later mapped by the server partition's physical device drivers into the physical I/O DMA address space of the physical I/O adapters using the kernel services, that have been similarly extended to call the H_PUT_RTCE hcall(), based upon the value of the LIOBN field reference by the cross memory descriptor. At this point, the server partition's VSCSI device driver delivers what appears to be a SCSI request to be decoded and routed through the server partition's file sub-system for processing. When the request completes, the server partition's VSCSI device driver is called by the file sub-system and it packages the summary status along with the request correlator into a response message that it sends to the client partition using the H_SEND_CRQ hcall(), then recycles the resources recorded in the I/O request control block, and the block itself.

The LIOBN value in the second window pane of the server partition's **"ibm,my-dma-window"** property is intended to be an indirect reference to the RTCE table of the client partition. If, for some reason, the physical location of the client partition's RTCE table changes or it becomes invalid, this level of indirection allows the hypervisor to determine the current target without changing the LIOBN number as seen by the server partition. The H_PUT_TCE and H_PUT_RTCE hcall()s do not map server partition memory into the second window pane; the second window pane is only available for use by server partition via the Logical RDMA services to reference memory mapped into it by the client partition's IOA.

This architecture does not specify the payload format of the requests or responses. However, the architectural intent is supplied in the following tables for reference.

Table 209. General Form of Reliable CRQ Element

Byte Offset	Field Name	Subfield Name	Description
0	Header		Contains Element Valid Bit plus Event Type Encodings (see Table 230, "CRQ Entry Header Byte Values," on page 621).
1	Payload	Format/Transport Event Code	For Valid Command Response Entries, see Table 210, "Example Reliable CRQ Entry Format Byte Definitions for VSCSI," on page 577. For Transport Event Codes see Table 232, "Transport Event Codes," on page 622.
2-15			Format Dependent.

Table 210. Example Reliable CRQ Entry Format Byte Definitions for VSCSI

Format Byte Value	Definition
0x0	Unused
0x1	VSCSI Requests
0x2	VSCSI Responses
0x03 - 0xFE	Reserved
0xFF	Reserved for Expansion

Table 211. Example VSCSI Command Queue Element

Byte Offset	Field Value	Description
0	0x80	Valid Header
1	0x01	VSCSI Request Format
2-3	NA	Reserved
4-7		Length of the request block to be transferred
8-15		I/O address of beginning of request

Table 212. Example VSCSI Response Queue Element

Byte Offset	Field Value	Description
0	0x80	Valid Header
1	0x02	VSCSI Response Format
2-3	NA	Reserved
4-7		Summary Status
8-15		8 byte command correlator

See also Appendix E, “A Protocol for VSCSI Communications,” on page 795.

16.5.2 Virtual SCSI Requirements

This normative section provides the general requirements for the support of VSCSI.

R1–16.5.2–1. For the VSCSI option: The platform must implement the Reliable Command/Response Transport option as defined in Section 17.2.3.1, “Reliable Command/Response Transport Option,” on page 637.

R1–16.5.2–2. For the VSCSI option: The platform must implement the Logical Remote DMA option as defined in Section 17.2.3.2, “Logical Remote DMA (LRDMA) Option,” on page 642.

In addition to the firmware primitives, and the structures they define, the partition’s OS needs to know specific information regarding the configuration of the virtual IOA’s that it has been assigned so that it can load and configure the correct device driver code. This information is provided by the OF device tree node associated with the virtual IOA (see Section 16.5.2.1, “Client Partition Virtual SCSI Device Tree Node,” on page 578 and Section 16.5.2.2, “Server Partition Virtual SCSI Device Tree Node,” on page 580).

16.5.2.1 Client Partition Virtual SCSI Device Tree Node

Client partition VSCSI device tree nodes have associated packages such as disk-label, deblocker, iso-13346-files and iso-9660-files as well as children nodes such as block and byte as appropriate to the specific virtual IOA configuration as would the node for a physical IOA of type scsi-3.

R1–16.5.2.1–1. For the VSCSI option: The platform's OF device tree for client partitions must include as a child of the **/vdevice** node, a node of name "v-scsi" as the parent of a sub-tree representing the virtual IOAs assigned to the partition.

R1–16.5.2.1–2. For the VSCSI option: The platform's **v-scsi** OF node must contain properties as defined in Table 214, "Properties of the VSCSI Node in the Server Partition," on page 580 (other standard I/O adapter properties are permissible as appropriate).

Table 213. Properties of the VSCSI Node in the Client Partition

Property Name	Required?	Definition
"name"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the virtual device name, the value shall be "v-scsi".
"device_type"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the virtual device type, the value shall be "vscsi".
"model"	NA	Property not present.
"compatible"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the programming models that are compatible with this virtual IOA, the value shall include **"IBM,v-scsi"**. "IBM,v-scsi-2" precedes "IBM,vsci" if it is included in the value of this property.
"used-by-rtas"	See Definition Column	Present if appropriate.
"ibm,loc-code"	Y	Property name specifying the unique and persistent location code associated with this virtual IOA presented as an encoded array as with **encode-string**. The value shall be of the form specified in Section 12.3.2.16, "Virtual Card Connector Location Codes," on page 335.
"reg"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the register addresses, used as the unit address (unit ID), associated with this virtual IOA presented as an encoded array as with **encode-phys** of length **"#address-cells"** value shall be 0xwhatever (virtual **"reg"** property used for unit address no actual locations used, therefore, the size field has zero cells (does not exist) as determined by the value of the **"#size-cells"** property).
"ibm,my-dma-window"	Y	Property name specifying the DMA window associated with this virtual IOA presented as an encoded array of three values (LIOBN, phys, size) encoded as with **encode-int**, **encode-phys**, and **encode-int**.
"interrupts"	Y	Standard property name specifying the interrupt source number and sense code associated with this virtual IOA presented as an encoded array of two cells encoded as with **encode-int** with the first cell containing the interrupt source number, and the second cell containing the sense code 0 indicating positive edge triggered. The interrupt source number being the value returned by the H_XIRR or H_IPOLL hcall().
"ibm,my-drc-index"	For DR	Present if the platform implements DR for this node.
"ibm,#dma-size-cells"	See Definition Column	Property name, to define the package's dma address size format. The property value specifies the number of cells that are used to encode the size field of dma-window properties. This property is present when the dma address size format cannot be derived using the method described in the definition for the **"ibm,#dma-size-cells"** property in Appendix B, "LoPAPR Binding," on page 661.
"ibm,#dma-address-cells"	See Definition Column	Property name, to define the package's dma address format. The property value specifies the number of cells that are used to encode the physical address field of dma-window properties. This property is present when the dma address format cannot be derived using the method described in the definition for the **"ibm,#dma-address-cells"** property in Appendix B, "LoPAPR Binding," on page 661.

R1–16.5.2.1–3. For the VSCSI option: The platform's **v-scsi** node must have as children the appropriate block (**disk**) and byte (**tape**) nodes.

16.5.2.2 Server Partition Virtual SCSI Device Tree Node

Server partition VSCSI IOA nodes have no children nodes.

R1–16.5.2.2–1. For the VSCSI option: The platform's OF device tree for server partitions must include as a child of the **/vdevice** node, a node of name "v-scsi-host" as the parent of a sub-tree representing the virtual IOAs assigned to the partition.

R1–16.5.2.2–2. For the VSCSI option: The platform's **v-scsi-host** node must contain properties as defined in Table 214, "Properties of the VSCSI Node in the Server Partition," on page 580 (other standard I/O adapter properties are permissible as appropriate).

Table 214. Properties of the VSCSI Node in the Server Partition

Property Name	Required?	Definition
"name"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the virtual device name, the value shall be "v-scsi-host".
"device_type"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the virtual device type, the value shall be "v-scsi-host".
"model"	NA	Property not present.
"compatible"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the programming models that are compatible with this virtual IOA, the value shall include "IBM,v-scsi-host". "IBM,v-scsi-host-2" precedes "IBM,vsci-host" if it is included in the value of this property.
"used-by-rtas"	See Definition Column	Present if appropriate.
"ibm,loc-code"	Y	Property name specifying the unique and persistent location code associated with this virtual IOA presented as an encoded array as with **encode-string**. The value shall be of the form Section 12.3.2.16, "Virtual Card Connector Location Codes," on page 335.
"reg"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the register addresses, used as the unit address (unit ID), associated with this virtual IOA presented as an encoded array as with **encode-phys** of length **"#address-cells"** value shall be 0xwhatever (virtual **"reg"** property used for unit address no actual locations used, therefore, the size field has zero cells (does not exist) as determined by the value of the **"#size-cells"** property).
"ibm,my-dma-window"	Y	Property name specifying the DMA window associated with this virtual IOA presented as an encoded array of two sets (two window panes) of three values (LIOBN, phys, size) encoded as with **encode-int**, **encode-phys**, and **encode-int**. Of these two triples, the first describes the window pane used to map server partition memory, the second is the window pane through which the client partition maps its memory that it makes available to the server partition. (Note the mapping between the LIOBN in the second window pane of a server virtual IOA's **"ibm,my-dma-window"** property and the corresponding client IOA's RTCE table is made when the CRQ successfully completes registration. See Section 17.2.1.2, "RTCE Table and Properties of the Children of the /vdevice Node," on page 601 for more information on window panes.)
"interrupts"	Y	Standard property name specifying the interrupt source number and sense code associated with this virtual IOA presented as an encoded array of two cells encoded as with **encode-int** with the first cell containing the interrupt source number, and the second cell containing the sense code 0 indicating positive edge triggered. The interrupt source number being the value returned by the H_XIRR or H_IPOLL hcall()

Table 214. Properties of the VSCSI Node in the Server Partition

Property Name	Required?	Definition
"ibm,my-drc-index"	For DR	Present if the platform implements DR for this node.
"ibm,vserver"	Y	Property name specifying that this is a virtual server node.
"ibm,#dma-size-cells"	See Definition Column	Property name, to define the package's dma address size format. The property value specifies the number of cells that are used to encode the size field of dma-window properties. This property is present when the dma address size format cannot be derived using the method described in the definition for the **"ibm,#dma-size-cells"** property in Appendix B, "LoPAPR Binding," on page 661.
"ibm,#dma-address-cells"	See Definition Column	Property name, to define the package's dma address format. The property value specifies the number of cells that are used to encode the physical address field of dma-window properties. This property is present when the dma address format cannot be derived using the method described in the definition for the **"ibm,#dma-address-cells"** property in Appendix B, "LoPAPR Binding," on page 661.

16.6 Virtual Terminal (Vterm)

This section defines the Virtual Terminal (Vterm) options (Client Vterm option and Server Vterm option). Vterm IOAs are of the hypervisor simulated class of VIO. See also Appendix D, “A Protocol for a Virtual TTY Interface,” on page 787.

16.6.1 Vterm General

This section contains an informative outline of the architectural intent of the use of Vterm support.

The architectural metaphor for the Vterm IOA is that of an Async IOA. The connection at the other end of the Async “cable” may be another Vterm IOA in a server partition, the hypervisor, or the HMC.

A partition’s device tree contains one or more nodes notifying the partition that it has been assigned one or more Vterm client adapters (each LoPAPR partition has at least one). The node’s **“type”** and **“compatible”** properties notify the partition that the virtual adapter is a Vterm client adapter. The **unit address** of the node is used by the partition to map the virtual device(s) to the OS’s corresponding logical representations. The node’s **“interrupts”** property, if it exists, specifies the interrupt source number that has been assigned to the client Vterm IOA for receive data. The partition, uses the H_GET_TERM_CHAR and H_PUT_TERM_CHAR hcall()s to receive data from and send data to the client Vterm IOA. If the node contains the **“interrupts”** property, the partition may optionally use the *ibm,int-on*, *ibm,int-off*, *ibm,set-xive, ibm,get-xive* RTAS calls, and the H_VIO_SIGNAL hcall() to manage the client Vterm IOA interrupt.

A partition’s device tree may also contain one or more node(s) notifying the partition that it is requested to supply server Vterm IOA services for one or more client Vterm IOAs. The node’s **“type”** and **“compatible”** properties notify the partition that the virtual adapter is a server Vterm IOA. The unit address (**Unit ID)** of the node is used by the partition to map the virtual device(s) to the OS’s corresponding logical representations. The node’s **“interrupts”** property specifies the interrupt source number that has been assigned to the server Vterm IOA for receive data. The partition uses the H_VTERM_PARTNER_INFO hcall() to find out which unit address(es) in which partition(s) to which it is allowed to attach (that is, to which client Vterm IOAs it is allowed to attach). The partition then uses the H_REGISTER_VTERM to setup the connection between a server and a client Vterm IOAs, and uses the H_GET_TERM_CHAR and H_PUT_TERM_CHAR hcall()s to receive data from and send data to the server Vterm IOA. In addition, the partition may optionally use the *ibm,int-on*, *ibm,int-off*, *ibm,set-xive, ibm,get-xive* RTAS calls, and the H_VIO_SIGNAL hcall() to manage the server Vterm IOA interrupt.

Table 215, “Client Vterm versus Server Vterm Comparison,” on page 582 shows a comparison between the client and server versions of Vterm.

Table 215. Client Vterm versus Server Vterm Comparison

Client	Server
The following hcall()s apply: H_PUT_TERM_CHAR H_GET_TERM_CHAR H_VIO_SIGNAL (optional use with Client)	
N/A	The following hcall()s are valid: H_VTERM_PARTNER_INFO H_REGISTER_VTERM H_FREE_VTERM
vty node	**vty-server** node

Table 215. Client Vterm versus Server Vterm Comparison *(Continued)*

Client	Server
The **"reg"** property or the **vty** node(s) enumerates the valid client Vterm IOA unit address(es)	The **"reg"** property or the **vty-server** node(s) enumerates the valid server Vterm IOA unit address(es) H_VTERM_PARTNER_INFO is used to get valid client Vterm IOA partition ID(s) and corresponding unit address(es) to which the server Vterm IOA is allowed to connect
"interrupts" property optional: Platform may or may not provide If provided, Vterm driver may or may not use	**"interrupts"** property required: Platform must provide If provided, Vterm driver may or may not use

16.6.2 Vterm Requirements

This normative section provides the general requirements for the support of Vterm.

R1–16.6.2–1. For the LPAR option: the Client Vterm option must be implemented.

16.6.2.1 Character Put and Get hcall()s

The following hcall()s are used to send data to and get data from both the client and sever Vterm IOAs.

16.6.2.1.1 H_GET_TERM_CHAR

Syntax:

uint64 hcall(const uint64 H_GET_TERM_CHAR, int64 termno)

Parameters:

- termno: The unit address of the Vterm IOA, from the **"reg"** property of the Vterm IOA.

Semantics:

- Hypervisor checks the termno parameter for validity against the Vterm IOA unit addresses assigned to the partition, else return H_Parameter.
- Hypervisor returns H_Hardware if it detects that the virtual console terminal physical connection is not working.
- Hypervisor returns H_Closed if it detects that the virtual console associated with the termno parameter is not open (in the case of connection to a server Vterm IOA, this means that the server code has not made the connection to this specific client Vterm IOA).
- Hypervisor returns H_Success in all other cases, returning maximum number of characters available in the partition's virtual console terminal input buffer (up to 16) -- a len value of 0 indicates that the input buffer is empty.
- Upon return with H_Success register R4 contains the number of bytes (if any) returned in registers R5 and R6.
- Upon return with H_Success the return character string starts in the high order byte of register R5 and proceeds toward the low order byte in register R6 for the number of characters specified in R4. The contents of all other byte locations of registers R5 and R6 are undefined.

16.6.2.1.2 H_PUT_TERM_CHAR

Syntax:

int64 hcall(const uint64 H_PUT_TERM_CHAR, int64 termno, int64 len, unit64 char0_7, init64 char8_15)

Parameters:

- termno: The unit address of the Vterm IOA, from the **"reg"** property of the Vterm IOA.
- len: The length of the string to transmit through the virtual terminal port. Valid values are in the range of 0-16.
- char0_7 and char8_15: The string starts in the high order byte of register R6 and proceeds toward the low order byte in register R7

Semantics:

- Hypervisor checks the termno parameter for validity against the Vterm IOA unit addresses assigned to the partition, else return H_Parameter.
- Hypervisor returns H_Hardware if it detects that the virtual console terminal physical connection is not working.
- Hypervisor returns H_Closed if it detects that the virtual console session is not open (in the case of connection to a server Vterm IOA, this means that the server code has not made the connection to this specific client Vterm IOA).
- If the length parameter is outside of the values 0-16 the hypervisor immediately returns H_Parameter with no other action.
- If the partition's virtual console terminal buffer has room for the entire string, the hypervisor queues the output string and returns H_Success. Note: There is always room for a zero length string (a zero length write can be used to test the virtual console terminal connection).
- If the buffer cannot hold the entire string, no data is enqueued and the return code is H_Busy.

16.6.2.2 Interrupts

The interrupt source number is presented in the **"interrupts"** property of the **Vterm** node, when receive queue interrupts are implemented for the Vterm. The *ibm,int-on*, *ibm,int-off*, *ibm,set-xive, ibm,get-xive* RTAS calls, and H_VIO_SIGNAL hcall() are used to manage the interrupt.

Interrupts and the **"interrupts"** property are always implemented for the server Vterm IOA, and may be implemented for the client Vterm IOA.

The interrupt mechanism is edge-triggered and is capable of presenting only one interrupt signal at a time from any given interrupt source. Therefore, no additional interrupts from a given source are ever signaled until the previous interrupt has been processed through to the issuance of an H_EOI hcall(). Specifically, even if the interrupt mode is enabled, the effect is to interrupt on an empty to non-empty transition of the receiver queue or upon the closing of the connection between the server and client. However, as with any asynchronous posting operation race conditions are to be expected. That is, an enqueue can happen in a window around the H_EOI hcall() so that the receiver should poll the receive queue after an H_EOI using H_GET_TERM_CHAR after an H_EOI, to prevent losing initiative.

R1–16.6.2.2–1. For the Server Vterm option: The platform must implement the **"interrupts"** property in all server **Vterm** device tree nodes (**vty-server**), and must set the interrupt in that property for the receive data interrupt for the IOA.

R1–16.6.2.2–2. For the Client Vterm and Server Vterm options: When implemented, the characteristics of the Vterm interrupts must be as follows:

a. All must be edge-triggered.

b. The receive interrupt must be activated when the Vterm receive queue goes from empty to non-empty.

c. The receive interrupt must be activated when the Vterm connection from the client to the server goes from open to closed.

16.6.2.3 Client Vterm Device Tree Node (vty)

All platforms that implement LPAR, also implement at least one client Vterm IOA per partition.

R1–16.6.2.3–1. For the Client Vterm option: The H_GET_TERM_CHAR and H_TERM_CHAR hcall()s must be implemented.

R1–16.6.2.3–2. For the Client Vterm option: The platform's OF device tree must include as a child of the **/vdevice** node, one or more nodes of type "vty"; one for each client Vterm IOA.

R1–16.6.2.3–3. For the Client Vterm option: The platform's **vty** OF node must contain properties as defined in Table 216, "Properties of the vty Node (Client Vterm IOA)," on page 585 (other standard I/O adapter properties are permissible as appropriate).

Table 216. Properties of the **vty** Node (Client Vterm IOA)

Property Name	Required?	Definition
"name"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the virtual device name. The value shall be "vty".
"device_type"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the virtual device type. The value shall be "serial".
"model"	NA	Property not present.
"compatible"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the programming models that are compatible with this virtual IOA. The value shall include "hvterm1" when the virtual IOA will connect to a server with no special protocol, and shall include "hvterm-protocol" when the virtual IOA will connect to a server that requires a protocol to control modems or hardware control signals.
"used-by-rtas"	NA	Property not present.
"ibm,loc-code"	Y	Property name specifying the unique and persistent location code associated with this virtual IOA presented as an encoded array as with **encode-string**. The value shall be of the form specified in Section 12.3.2.16, "Virtual Card Connector Location Codes," on page 335.
"reg"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the register addresses, used as the unit address (unit ID), associated with this virtual IOA presented as an encoded array as with **encode-phys** of length **"#address-cells"** value shall be 0xwhatever (virtual **"reg"** property used for unit address no actual locations used, therefore, the size field has zero cells (does not exist) as determined by the value of the **"#size-cells"** property).
"interrupts"	See Definition Column	Standard property name specifying the interrupt source number and sense code associated with this virtual IOA presented as an encoded array of two cells encoded as with **encode-int** with the first cell containing the interrupt source number, and the second cell containing the sense code 0 indicating positive edge triggered. The interrupt source number being the value returned by the H_XIRR or H_IPOLL hcall(). If provided, this property will present one interrupt; the receive data interrupt.
"ibm,my-drc-index"	For DR	Present if the platform implements DR for this node.

R1–16.6.2.3–4. For the Client Vterm option: If the compatible property in the **vty** node is **"hvterm-protocol"**, then the protocol that the client must use is defined in the document entitled *Protocol for Support of Physical Serial Port Using a Virtual TTY Interface*.

16.6.2.4 Server Vterm

Server Vterm IOAs allow a partition to serve a partner partition's client Vterm IOA.

16.6.2.4.1 Server Vterm Device Tree Node (vty-server) and Other Requirements

R1–16.6.2.4.1–1. For the Server Vterm option: The H_GET_TERM_CHAR, H_PUT_TERM_CHAR, H_VTERM_PARTNER_INFO, H_REGISTER_VTERM, and H_FREE_VTERM hcall()s must be implemented.

R1–16.6.2.4.1–2. For the Server Vterm option: The platform's OF device tree for partitions implementing server Vterm IOAs must include as a child of the **/vdevice** node, one or more nodes of type "vty-server"; one for each server Vterm IOA.

R1–16.6.2.4.1–3. For the Server Vterm option: The platform's **vty-server** node must contain properties as defined in Table 217, "Properties of the vty-server Node (Server Vterm IOA)," on page 586 (other standard I/O adapter properties are permissible as appropriate).

Table 217. Properties of the **vty-server** Node (Server Vterm IOA)

Property Name	Required?	Definition
"name"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the virtual device name. The value shall be "vty-server".
"device_type"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the virtual device type. The value shall be "serial-server".
"model"	NA	Property not present.
"compatible"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the programming models that are compatible with this virtual IOA. The value shall include "hvterm2".
"used-by-rtas"	NA	Property not present.
"ibm,loc-code"	Y	Property name specifying the unique and persistent location code associated with this virtual IOA presented as an encoded array as with **encode-string**. The value shall be of the form Section 12.3.2.16, "Virtual Card Connector Location Codes," on page 335.
"reg"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the register addresses, used as the unit address (unit ID), associated with this virtual IOA presented as an encoded array as with **encode-phys** of length specified by **"#address-cells"** value shall be 0xwhatever (virtual **"reg"** property used for unit address no actual locations used, therefore, the size field has zero cells (does not exist) as determined by the value of the **"#size-cells"** property).
"interrupts"	Y	Standard property name specifying the interrupt source number and sense code associated with this virtual IOA presented as an encoded array of two cells encoded as with **encode-int** with the first cell containing the interrupt source number, and the second cell containing the sense code 0 indicating positive edge triggered. The interrupt source number being the value returned by the H_XIRR or H_IPOLL hcall(). This property will present one interrupt; the receive data interrupt.
"ibm,my-drc-index"	For DR	Present if the platform implements DR for this node.

Table 217. Properties of the **vty-server** Node (Server Vterm IOA) ***(Continued)***

Property Name	Required?	Definition
"ibm,vserver"	Y	Property name specifying that this is a virtual server node.

16.6.2.4.2 Server Vterm hcall()s

The following hcall()s are unique to the server Vterm IOA.

16.6.2.4.2.1 H_VTERM_PARTNER_INFO

This hcall is used to retrieve the list of Vterms to which the specified server Vterm IOA is permitted to connect. The list is retrieved by making repeated calls, and returns sets of triples: partner partition ID, partner unit address, and partner location code. Passing in the previously returned value will return the next value in the list of allowed connections. Passing in a value of 0xFF...FF will return the first value in the list.

Syntax:

```
int64 hcall(    /* H_Success,              Expected Return Code
                H_Parameter,              One or more of the parameters are invalid
                H_Hardware                The function failed due to unrecoverable hardware failure. */

    const uint64 H_VTERM_PARTNER_INFO, /* Gets possible partner connections */
    uint64    unit-address,           /* As specified in the Virtual IOA's device tree node */
    uint64    partner-partition-id,   /* the last partner partition ID returned, or 0xFF...FF to start */
    uint64    partner-unit-addr       /* the last partner unit address returned, or 0xFF...FF to start */
    uint64    buffr-addr);            /* the logical address of the buffer where the data is to be returned */
```

Parameters:

- unit-address: Virtual IOA's unit address, as specified in the IOA's device tree node.
- partner-partition-id: The partner partition ID of the last partner partition ID and partner unit address pair returned. If a value of 0xFF...FF is specified, the call returns the first item in the list.
- partner-unit-addr: The partner unit address of the last partner partition ID and partner unit address pair returned. If a value of 0xFF...FF is specified, the call returns the first item in the list.
- buffr-addr: The logical address of a single page in memory, belonging to the calling partition, which is used to return the next triple of information (partner partition ID, partner unit address, and Converged Location Code). The calling partition cannot migrate the page during the duration of the call, otherwise the call will fail.

Buffer format on return with H_Success:

- First eight bytes: Eight byte partner partition ID of the partner partition ID and partner unit address pair from the list, or 0xFF...FF if partner partition ID and partner unit address passed in the input parameters was the last item in the list of valid connections.
- Second eight bytes: Eight byte partner unit address associated with the partner partition ID (as returned in first 8 bytes of the buffer), or 0xFF...FF if the partner partition ID and partner unit address passed in the input parameters was the last item in the list of valid connections.
- Beginning at the 17 byte in the buffer: NULL-terminated Converged Location Code associated with the partner unit address and partner partition ID (a returned in the first 16 bytes of the buffer), or just a NULL string if the partner partition ID and partner unit address passed in the input parameters was the last item in the list of valid connections.

Semantics:

- Validate that unit-address belongs to the partition and to a server Vterm IOA, else H_Parameter.
- If partner-partition-id and partner-unit-addr together do not match a valid partner partition ID and partner unit address pair in the list of valid connections for this unit-address, then return H_Parameter.
- If the 4 KB page associated with buffr-addr does not belong to the calling partition, then return H_Parameter.
- If the buffer associated with buffr-addr does not begin on a 4 K boundary, then return H_Parameter.
- If the calling partition attempts to migrate the buffer page associated with buffr-addr during the duration of the H_VTERM_PARTNER_INFO call, then return H_Parameter.
- If partner-partition-id is equal to 0xFF...FF, then select the first item from the list of valid connections, format the buffer as specified, above, for this item, and return H_Success.
- If partner-partition-id and partner-unit-addr together matches a valid partner partition ID and partner unit address pair in the list of valid connections, and if this is the last valid connection in the list, then format the buffer as specified, above, with the partner partition ID and partner unit address both set to 0xFF...FF, and the Converged Location Code set to the NULL string, and return H_Success.
- If partner-partition-id and partner-unit-addr together matches a valid partner partition ID and partner unit address pair in the list of valid connections, then select the next item from the list of valid connections, and format the buffer as specified, above, and return H_Success.

16.6.2.4.2.2 H_REGISTER_VTERM

This hcall has the effect of "opening" the connection to the client Vterm IOA in the specified partition ID and which has the specified unit address. The architectural metaphor for this is the connecting of the cable between two Async IOAs. The hcall fails if the partition does not have the authority to connect to the requested partition/unit address pair. The hcall() also fails if the specified partition/unit address is already in use (for example, by another partition or the HMC)

Syntax:

```
int64 hcall(    /* H_Success,              Expected Return Code
                H_Parameter,              One or more of the parameters are invalid
                H_Hardware                The function failed due to unrecoverable hardware failure. */

    const uint64 H_REGISTER_VTERM, /* Makes connection between server and partner Vterm IOAs */
    uint64   unit-address,         /* As specified in the Virtual IOA's device tree node */
    uint64   partner-partition-id, /* the partner ID to which to be connected */
    uint64   partner-unit-addr);   /* the partner unit address to which to be connected */
```

Parameters:

- unit-address: The server Virtual IOA's unit address, as specified in the IOA's device tree node.
- partner-partition-id: The partition ID of the partition ID and unit address pair to which to be connected.
- partner-unit-addr: The unit address of the partition ID and unit address pair to which to be connected.

Semantics:

- Validate that unit-address belongs to the partition and to a server Vterm IOA and that there does not exist a valid connection between this server Vterm IOA and a partner, else H_Parameter.

- If partner-partition-id and partner-unit-addr together do not match a valid partition ID and unit address pair in the list of valid connections for this unit-address, then return H_Parameter,
  - Else make connection between the server Vterm IOA specified by unit-address and the client Vterm IOA specified by the partner-partition-id and partner-unit-addr pair, allowing future H_PUT_TERM_CHAR and H_GET_TERM_CHAR operations to flow between the two Vterm IOAs, and return H_Success.

Software Implementation Note: An H_Parameter will be returned to the H_REGISTER_VTERM if a DLPAR operation has been performed which changes the list of possible server to client Vterm connections. After a DLPAR operation which affects a partition's server Vterm IOA connection list, a call to H_VTERM_PARTNER_INFO is needed to get the current list of possible connections.

16.6.2.4.2.3 H_FREE_VTERM

This hcall has the effect of "closing" the connection to the partition/unit address pair. The architectural metaphor for this is the removal of the cable between two Async IOAs. After closing, the partner partition's server Vterm IOA would now be available for serving by another partner (for example, another partition or the HMC).

Syntax:

```
int64 hcall(    /* H_Success,                Expected Return Code
                H_Parameter,                One or more of the parameters are invalid
                H_Hardware,                 The function failed due to unrecoverable hardware failure.
                H_Busy                      Try, again
                H_LongBusyOrder1mSec,       Try again (hint: may be up to 1mSec completion)
                H_LongBusyOrder10mSec,      Try again (hint: may be up to 10mSec completion) */
    const uint64 H_FREE_VTERM,              /* Break connection between server and partner Vterm IOAs */
    uint64      unit-address);              /* As specified in the Virtual IOA's device tree node */
```

Parameters:

- unit-address: Virtual IOA's unit address, as specified in the IOA's device tree node.

Semantics:

- Validate that the unit address belongs to the partition and to a server Vterm IOA and that there exists a valid connection between this server Vterm IOA and a partner, else H_Parameter.
- Break the connection between the server Vterm IOA specified by the unit address and the client Vterm IOA, preventing further H_PUT_TERM_CHAR and H_GET_TERM_CHAR operations between the two Vterm IOAs (until a future successful H_REGISTER_VTERM operation), and return H_Success.

Implementation Note: If the hypervisor returns an H_Busy, H_LongBusyOrder1mSec, or H_LongBusyOrder10mSec, software must call H_FREE_VTERM again with the same parameters. Software may choose to treat H_LongBusyOrder1mSec and H_LongBusyOrder10mSec the same as H_Busy. The hypervisor, prior to returning H_Busy, H_LongBusyOrder1mSec, or H_LongBusyOrder10mSec, will have placed the virtual adapter in a state that will cause it to not accept any new work nor surface any new virtual interrupts.

16.7 Virtual Fibre Channel (VFC) using NPIV

N_Port ID Virtualization (NPIV) is part of the Fibre Channel (FC) standards. NPIV allows multiple World Wide Port Names (WWPNs) to be mapped to a single physical port of a FC adapter. This section defines a Virtual Fibre Channel (VFC) interface to a server partition interfacing to a physical NPIV adapter that allows multiple partitions to share a physical port using different WWPNs. The implementation support is provided by code running in a server partition that uses the mechanisms of the Reliable Command/Response Transport and Logical Remote DMA of the Synchronous VIO Infrastructure to service I/O requests for code running in a client partition. The client partition appears to enjoy the services of its own FC adapter (see Section 17.2.3, “Partition Managed Class - Synchronous Infrastructure,” on page 637) with a WWPN visible to the FC fabric. The terms server and client partitions refer to platform partitions that are respectively servers and clients of requests, usually I/O operations, using the physical I/O adapters (IOAs) that are assigned to the server partition. This allows a platform to have more client partitions than it may have physical I/O adapters because the client partitions share I/O adapters via the server partition.

The VFC model makes use of Remote DMA which is built upon the architecture specified in the following sections:

- Section 17.2.1, “VIO Infrastructure - General,” on page 600
- Section 17.2.2, “Partition Managed Class Infrastructure - General,” on page 620
- Section 17.2.3, “Partition Managed Class - Synchronous Infrastructure,” on page 637

16.7.1 VFC and NPIV General

This section contains an informative outline of the architectural intent of the use of VFC and NIPV, and it assumes the user is familiar with Section 16.5.1, “VSCSI General,” on page 575 concerning VSCSI architecture and the with the FC standards. Other implementations of the server and client partition code, consistent with this architecture, are possible and may be preferable.

The client partition provides the virtual equivalent of a single port FC adapter via each VFC client IOA. The platform, through the partition definition, provides means for defining the set of virtual IOA’s owned by each partition and their respective location codes. The platform also provides, through partition definition, instructions to connect each client partition’s VFC client IOA to a specific server partition’s VFC server IOA. The mechanism for specifying this partition definition is beyond the scope of this architecture. The human readable handle associated with the partition definition of virtual IOAs and their associated interconnection and resource configuration is the virtual location code. The OF unit address (**`Unit ID)`** remains the invariant handle upon which the OS builds its “physical to logical” configuration. The platform also provides a method to assign unique WWPNs for each VFC client adapter. The port names are used by a SAN administrator to grant access to storage to a client partition. The mechanism for allocating port names is beyond the scope of this architecture.

The client partition’s device tree contains one or more nodes notifying the partition that it has been assigned one or more virtual adapters. The node’s **`“type”`** and **`“compatible”`** properties notify the partition that the virtual adapter is a VFC adapter. The **`unit address`** of the node is used by the client partition to map the virtual device(s) to the OS’s corresponding logical representations. The **`“ibm,my-dma-window”`** property communicates the size of the RTCE table window panes that the hypervisor has allocated. The node also specifies the interrupt source number that has been assigned to the Reliable Command/Response Transport connection and the RTCE range that the client partition device driver may use to map its memory for access by the server partition via Logical Remote DMA. The client partition also reads it's WWPNs from the device tree. Two WWPNs are presented to the client in the properties **`“ibm,port-wwn-1”`**, and **`“ibm,port-wwn-2”`**, and the server tells the client, through a CRQ protocol exchange, which one of the two to use. The client partition, uses the four hcall()s associated with the Reliable Command/Response Transport facility to register and deregister its CRQ, manage notification of responses, and send command requests to the server partition.

The server partition's device tree contains one or more node(s) notifying the partition that it is requested to supply VFC services for one or more client partitions. The unit address (**`Unit ID)`** of the node is used by the server partition to map to the local logical devices that are represented by this VFC device. The node also specifies the interrupt source number that has been assigned to the Reliable Command/Response Transport connection and the RTCE range that the server partition device driver may use for its copy Logical Remote DMA. The server partition uses the four hcall()s associated with the Reliable Command/Response Transport facility to register and deregister its Command request queue, manage notification of new requests, and send responses back to the client partition. In addition, the server partition uses the hcall()s of the Logical Remote DMA facility to manage the movement of commands and data associated with the client requests.

The client partition, upon noting the device tree entry for the virtual adapter, loads the device driver associated with the value of the **`"compatible"`** property. The device driver, when configured and opened, allocates memory for its CRQ (an array, large enough for all possible responses, of 16 byte elements), pins the queue and maps it into the I/O space of the RTCE window specified in the **`"ibm,my-dma-window"`** property using the standard kernel mapping services that subsequently use the H_PUT_TCE hcall(). The queue is then registered using the H_REG_CRQ hcall(). Next, I/O request control blocks (within which the I/O requests commands are built) are allocated, pinned, and mapped into I/O address space. Finally, the device driver registers to receive control when the interrupt source specified in the virtual IOA's device tree node signals.

Once the CRQ is setup, the device driver queues an Initialization Command/Response with the second byte of "Initialize" in order to attempt to tell the hosting side that everything is setup on the hosted side. The response to this send may be that the send has been dropped or has successfully been sent. If successful, the sender should expect back an Initialization Command/Response with a second byte of "Initialization Complete," at which time the communication path can be deemed to be open. If dropped, then the sender waits for the receipt of an Initialization Command/Response with a second byte of "Initialize," at which time an "Initialization Complete" message is sent, and if that message is sent successfully, then the communication path can be deemed to be open.

When the VFC Adapter device driver receives an I/O request from one of the FC device head drivers, it executes the following sequence. First an I/O request control block is allocated. Then it builds the FC Information Unit (FC IU) request within the control block, adds a correlator field (to be returned in the subsequent response), I/O maps any target memory buffers and places their DMA descriptors into the I/O request control block. With the request constructed in the I/O request control block, the driver constructs a DMA descriptor (Starting Offset, and length) representing the FC IU within the I/O request control block. It also constructs a DMA descriptor for the FC Response Unit. Lastly, the driver passes the I/O request's DMA descriptor to the server partition using the H_SEND_CRQ hcall(). Provided that the H_SEND_CRQ hcall() succeeds, the VFC Adapter device driver returns, waiting for the response interrupt indicating that a response has been posted by the server partition to the device driver's response queue. The response queue entry contains the summary status and request correlator. From the request correlator, the device driver accesses the I/O request control block, the summary status, and the FC Response Unit and determines how to complete the processing of the I/O request.

Notice that the client partition only uses the Reliable Command/Response Transport primitives; it does not use the Logical Remote DMA primitives. Since the server partition's RTCE tables are not authorized for access by the client partition, any attempt by the client partition to modify server partition memory would be prevented by the hypervisor. RTCE table access is granted on a connection by connection basis (client/server virtual device pair). If a client partition happens to be serving some other logical device, then the partition is entitled to use Logical Remote DMA for the virtual devices that is serving.

The server partition, upon noting the device tree entry for the virtual adapter, loads the device driver associated with the value of the **`"compatible"`** property. The device driver, when configured and opened, allocates memory for its request queue (an array, large enough for all possible outstanding requests, of 16 byte elements). The driver then pins the queue and maps it into I/O space, via the kernel's I/O mapping services that invoke the H_PUT_TCE hcall(), using the first window pane specified in the **`"ibm,my-dma-window"`** property. The queue is then registered using the H_REG_CRQ hcall(). Next, I/O request control blocks (within which the I/O request commands are built) are allocated, pinned, and I/O mapped. Finally the device driver registers to receive control when the interrupt source specified in the virtual IOA's device tree node signals.

Once the CRQ is setup, the device driver queues an Initialization Command/Response with the second byte of “Initialize” in order to attempt to tell the hosted side that everything is setup on the hosting side. The response to this send may be that the send has been dropped or has successfully been sent. If successful, the sender should expect back an Initialization Command/Response with a second byte of “Initialization Complete,” at which time the communication path can be deemed to be open. If dropped, then the sender waits for the receipt of an Initialization Command/Response with a second byte of “Initialize,” at which time an “Initialization Complete” message is sent, and if that message is sent successfully, then the communication path can be deemed to be open.

When the server partition’s device driver receives an I/O request from its corresponding client partition’s VFC adapter drivers, it is notified via the interrupt registered for above. The server partition’s device driver selects an I/O request control block for the requested operation. It then uses the DMA descriptor from the request queue element to transfer the FC IU request from the client partition’s I/O request control block to its own (allocated above), using the H_COPY_RDMA hcall() through the second window pane specified in the **“ibm,my-dma-window”** property. The server partition’s device driver then uses kernel services, that are extended, to register the I/O request’s DMA descriptors into extended capacity cross memory descriptors (ones capable of recording the DMA descriptors). These cross memory descriptors are later mapped by the server partition’s physical device drivers into the physical I/O DMA address space of the physical I/O adapters using the kernel services, that have been similarly extended to call the H_PUT_RTCE hcall(), based upon the value of the LIOBN field reference by the cross memory descriptor. At this point, the server partition’s VFC device driver delivers what appears to be a FC IU request to be routed through the server partition’s adapter driver. When the request completes, the server partition’s VFC device driver is called through a registered entry point and it packages the summary status along with the request correlator into a response message that it sends to the client partition using the H_SEND_CRQ hcall(), then recycles the resources recorded in the I/O request control block, and the block itself.

The LIOBN value in the second window pane of the server partition’s **“ibm,my-dma-window”** property is intended to be an indirect reference to the RTCE table of the client partition. If, for some reason, the physical location of the client partition’s RTCE table changes or it becomes invalid, this level of indirection allows the hypervisor to determine the current target without changing the LIOBN number as seen by the server partition. The H_PUT_TCE and H_PUT_RTCE hcall()s do not map server partition memory into the second window pane; the second window pane is only available for use by server partition via the Logical RDMA services to reference memory mapped into it by the client partition’s IOA.

This architecture does not specify the payload format of the requests or responses. However, the architectural intent is supplied in the following tables for reference.

Table 218. General Form of Reliable CRQ Element

Byte Offset	Field Name	Subfield Name	Description
0	Header		Contains Element Valid Bit plus Event Type Encodings (see Table 230, “CRQ Entry Header Byte Values,” on page 621).
1	Payload	Format/Transport Event Code	For Valid Command Response Entries, see Table 219, “Example Reliable CRQ Entry Format Byte Definitions for VFC,” on page 592. For Transport Event Codes see Table 232, “Transport Event Codes,” on page 622.
2-15			Format Dependent.

Table 219. Example Reliable CRQ Entry Format Byte Definitions for VFC

Format Byte Value	Definition
0x0	Unused

Table 219. Example Reliable CRQ Entry Format Byte Definitions for VFC

Format Byte Value	Definition
0x01	VFC Requests
0x02 - 0x03	Reserved
0x04	Management Datagram
0x05 - 0xFE	Reserved
0xFF	Reserved for Expansion

Table 220. Example VFC Command Queue Element

Byte Offset	Field Value	Description
0	0x80	Valid Header
1	0x01	VFC Requests
1	0x04	Management Datagram
2-3	NA	Reserved
4-7		Length of the request block to be transferred
8-15		I/O address of beginning of request

Table 221. Example VFC Response Queue Element

Byte Offset	Field Value	Description
0	0x80	Valid Header
1	0x01	VFC Response Format
1	0x02	Asynchronous Event
1	0x04	Management Datagram
2-3	NA	Reserved
4-7		Summary Status
8-15		8 byte command correlator

16.7.2 VFC and NPIV Requirements

This normative section provides the general requirements for the support of VFC.

R1–16.7.2–1. For the VFC option: The platform must implement the Reliable Command/Response Transport option as defined in Section 17.2.3.1, “Reliable Command/Response Transport Option,” on page 637.

R1–16.7.2–2. For the VFC option: The platform must implement the Logical Remote DMA option as defined in Section 17.2.3.2, “Logical Remote DMA (LRDMA) Option,” on page 642.

R1–16.7.2–3. For the VFC option: The platform must allocate a WWPN pair for each VFC client and must present the WWPNs to the VFC clients in their OF device tree Section 222, “Properties of the VFC Node in the Client Partition,” on page 594.

In addition to the firmware primitives, and the structures they define, the partition’s OS needs to know specific information regarding the configuration of the virtual IOA’s that it has been assigned so that it can load and configure the correct device driver code. This information is provided by the OF device tree node associated with the virtual IOA (see Section 16.7.2.1, “Client Partition VFC Device Tree Node,” on page 594 and Section 16.7.2.2, “Server Partition VFC Device Tree Node,” on page 595).

16.7.2.1 Client Partition VFC Device Tree Node

Client partition VFC device tree nodes have associated packages such as disk-label, deblocker, iso-13346-files and iso-9660-files as well as children nodes such as block and byte as appropriate to the specific virtual IOA configuration as would the node for a physical FC IOA.

R1–16.7.2.1–1. For the VFC option: The platform’s OF device tree for client partitions must include as a child of the **`/vdevice`** node, a node of name **`"vfc-client"`** as the parent of a sub-tree representing the virtual IOAs assigned to the partition.

R1–16.7.2.1–2. For the VFC option: The platform’s **`vfc-client`** OF node must contain properties as defined in Table 222, “Properties of the VFC Node in the Client Partition,” on page 594 (other standard I/O adapter properties are permissible as appropriate).

Table 222. Properties of the VFC Node in the Client Partition

Property Name	Required?	Definition
`"name"`	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the virtual device name, the value shall be `"vfc-client"`.
`"device_type"`	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the virtual device type, the value shall be `"fcp"`.
`"model"`	NA	Property not present.
`"compatible"`	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the programming models that are compatible with this virtual IOA, the value shall include **`"IBM,vfc-client"`**.
`"used-by-rtas"`	See Definition Column	Present if appropriate.
`"ibm,loc-code"`	Y	Property name specifying the unique and persistent location code associated with this virtual IOA presented as an encoded array as with **`encode-string`**. The value shall be of the form specified in Section 12.3.2.16, “Virtual Card Connector Location Codes,” on page 335.
`"reg"`	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the register addresses, used as the unit address (unit ID), associated with this virtual IOA presented as an encoded array as with **`encode-phys`** of length **`"#address-cells"`** value shall be 0xwhatever (virtual **`"reg"`** property used for unit address no actual locations used, therefore, the size field has zero cells (does not exist) as determined by the value of the **`"#size-cells"`** property).
`"ibm,my-dma-window"`	Y	Property name specifying the DMA window associated with this virtual IOA presented as an encoded array of three values (LIOBN, phys, size) encoded as with **`encode-int`**, **`encode-phys`**, and **`encode-int`**.

Table 222. Properties of the VFC Node in the Client Partition

Property Name	Required?	Definition
"interrupts"	Y	Standard property name specifying the interrupt source number and sense code associated with this virtual IOA presented as an encoded array of two cells encoded as with **encode-int** with the first cell containing the interrupt source number, and the second cell containing the sense code 0 indicating positive edge triggered. The interrupt source number being the value returned by the H_XIRR or H_IPOLL hcall().
"ibm,my-drc-index"	For DR	Present if the platform implements DR for this node.
"ibm,#dma-size-cells"	See Definition Column	Property name, to define the package's dma address size format. The property value specifies the number of cells that are used to encode the size field of dma-window properties. This property is present when the dma address size format cannot be derived using the method described in the definition for the **"ibm,#dma-size-cells"** property in Appendix B, "LoPAPR Binding," on page 661.
"ibm,#dma-address-cells"	See Definition Column	Property name, to define the package's dma address format. The property value specifies the number of cells that are used to encode the physical address field of dma-window properties. This property is present when the dma address format cannot be derived using the method described in the definition for the **"ibm,#dma-address-cells"** property in Appendix B, "LoPAPR Binding," on page 661.
"ibm,port-wwn-1"	See Definition Column	Property that represents one of two WWPNs assigned to this VFC client node. This property is a **prop-encoded-array** each encoded with **encode-int**. The array consists of the high order 32 bits and low order 32 bits of the WWPN such that (32 bits high | 32 bits low) is the 64 bit WWPN. The WWPN that the client is to use (**"ibm,port-wwn-1"** or **"ibm,port-wwn-2"**) is communicated to the client by the server as part of the client-server communications protocol.
"ibm,port-wwn-2"	See Definition Column	Property that represents one of two WWPNs assigned to this VFC client node This property is a **prop-encoded-array** each encoded with **encode-int**. The array consists of the high order 32 bits and low order 32 bits of the WWPN such that (32 bits high | 32 bits low) is the 64 bit WWPN. The WWPN that the client is to use (**"ibm,port-wwn-1"** or **"ibm,port-wwn-2"**) is communicated to the client by the server as part of the client-server communications protocol.

R1–16.7.2.1–3. For the VFC option: The platform's **vfc-client** node must have as children the appropriate block (**disk**) and byte (**tape**) nodes.

16.7.2.2 Server Partition VFC Device Tree Node

Server partition VFC IOA nodes have no children nodes.

R1–16.7.2.2–1. For the VFC option: The platform's OF device tree for server partitions must include as a child of the **/vdevice** node, a node of name "vfc-server" as the parent of a sub-tree representing the virtual IOAs assigned to the partition.

R1–16.7.2.2–2. For the VFC option: The platform's **vfc-server** node must contain properties as defined in Table 223, "Properties of the VFC Node in the Server Partition," on page 595 (other standard I/O adapter properties are permissible as appropriate).

Table 223. Properties of the VFC Node in the Server Partition

Property Name	Required?	Definition
"name"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the virtual device name, the value shall be "vfc-server".
"device_type"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the virtual device type, the value shall be "fcp".

Table 223. Properties of the VFC Node in the Server Partition

Property Name	Required?	Definition
"model"	NA	Property not present.
"compatible"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the programming models that are compatible with this virtual IOA, the value shall include "IBM,vfc-server".
"used-by-rtas"	See Definition Column	Present if appropriate.
"ibm,loc-code"	Y	Property name specifying the unique and persistent location code associated with this virtual IOA presented as an encoded array as with **encode-string**. The value shall be of the form Section 12.3.2.16, "Virtual Card Connector Location Codes," on page 335.
"reg"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the register addresses, used as the unit address (unit ID), associated with this virtual IOA presented as an encoded array as with **encode-phys** of length **"#address-cells"** value shall be 0xwhatever (virtual **"reg"** property used for unit address no actual locations used, therefore, the size field has zero cells (does not exist) as determined by the value of the **"#size-cells"** property).
"ibm,my-dma-window"	Y	Property name specifying the DMA window associated with this virtual IOA presented as an encoded array of two sets (two window panes) of three values (LIOBN, phys, size) encoded as with **encode-int**, **encode-phys**, and **encode-int**. Of these two triples, the first describes the window pane used to map server partition memory, the second is the window pane through which the client partition maps its memory that it makes available to the server partition. (Note the mapping between the LIOBN in the second window pane of a server virtual IOA's **"ibm,my-dma-window"** property and the corresponding client IOA's RTCE table is made when the CRQ successfully completes registration. See Section 17.2.1.2, "RTCE Table and Properties of the Children of the /vdevice Node," on page 601 for more information on window panes.)
"interrupts"	Y	Standard property name specifying the interrupt source number and sense code associated with this virtual IOA presented as an encoded array of two cells encoded as with **encode-int** with the first cell containing the interrupt source number, and the second cell containing the sense code 0 indicating positive edge triggered. The interrupt source number being the value returned by the H_XIRR or H_IPOLL hcall()
"ibm,my-drc-index"	For DR	Present if the platform implements DR for this node.
"ibm,vserver"	Y	Property name specifying that this is a virtual server node.
"ibm,#dma-size-cells"	See Definition Column	Property name, to define the package's dma address size format. The property value specifies the number of cells that are used to encode the size field of dma-window properties. This property is present when the dma address size format cannot be derived using the method described in the definition for the **"ibm,#dma-size-cells"** property in Appendix B, "LoPAPR Binding," on page 661.
"ibm,#dma-address-cells"	See Definition Column	Property name, to define the package's dma address format. The property value specifies the number of cells that are used to encode the physical address field of dma-window properties. This property is present when the dma address format cannot be derived using the method described in the definition for the **"ibm,#dma-address-cells"** property in Appendix B, "LoPAPR Binding," on page 661.

17 Virtualized Input/Output

Virtualized I/O is an optional feature of platforms that have hypervisor support. Virtual I/O (VIO) provides to a given partition the appearance of I/O adapters that do not have a one to one correspondence with a physical IOA. The hypervisor uses one of three techniques to realize a virtual IOA:

1. In the *hypervisor simulated* class, the hypervisor may totally simulate the adapter. For example, this is used in the virtual ethernet (IEEE VLAN) support (see Section 16.4, “Interpartition Logical LAN (ILLAN) Option,” on page 551). This technique is applicable to communications between partitions that are created by a single hypervisor instance.

2. In the *partition managed* class, a *server* partition provides the services of one of its IOA’s to a *partner* partition(s) (one or more *client* partitions[1] or one or more server partitions). In limited cases, a client may communicate directly to a client. A server partition provides support to interpret I/O requests from the partner partition, perform those requests on one or more of its devices, targeting the partner partition’s DMA buffer areas (for example, by using the Remote DMA (RDMA) facilities), and passing I/O responses back to the partner partition. For example, see Section 16.5, “Virtual SCSI (VSCSI),” on page 575.

3. In the *hypervisor managed* class, the hypervisor may provide low level hardware management (error and sub-channel allocation) so that partition level code may directly manage its assigned sub-channels.

This chapter is organized from general to specific. The overall structure of this architecture is as shown in Figure 33, “VIO Architecture Structure,” on page 597

{Figure marked to be hidden when Shark1+ conditional text is hidden}

{Figure marked to be visible when Shark1- conditional text is shown}

Figure 33. VIO Architecture Structure

17.1 Terminology used with VIO

Besides the general terminology defined on the first page of this chapter, Table 224, “Terminology used with VIO,” on page 597 will assist the reader in understanding the content of this chapter.

Table 224. Terminology used with VIO

Term	Definition
VIO	Virtual I/O. General term for all virtual I/O classes and virtual IOAs.
ILLAN	Interpartition Logical LAN. This option uses the hypervisor simulated class of virtual I/O to provide partition-to-partition LAN facilities without a real LAN IOA. See Section 16.4, “Interpartition Logical LAN (ILLAN) Option,” on page 551.

1.The term “hosted” is sometimes used for “client” and the term “hosting” is sometimes used for “server.” Note that a server IOA or partition can sometimes also be a client, and vice versa, so the terminology “client” and “server” tend to be less confusing than hosted and hosting.

Table 224. Terminology used with VIO *(Continued)*

Term	Definition
VSCSI	Virtual SCSI. This option provides the facilities for sharing physical SCSI type IOAs between partitions. Section 16.5, "Virtual SCSI (VSCSI)," on page 575.
Client Client VIO model	This terminology is mainly used with the partition managed class of VIO. The client, or client partition, is an entity which generally requests of a server partition, access to I/O to which it does not have direct access (that is, access to I/O which is under control of the server partition). Unlike the server, the client does not provide services to other partitions to share the I/O which resides in their partition. However, it possible to have the same partition be both a server and client partition, but under different virtual IOAs. The Client VIO model is one where the client partition maps part of its local memory into an RTCE table (as defined by the first window pane of the **"ibm,my-dma-window"** property), so that the server partition can get access to that client's local memory. An example of this is the VSCSI client (see Section 16.5, "Virtual SCSI (VSCSI)," on page 575 for more information).
Server Server VIO model	This terminology is mainly used with the partition managed class of VIO. The server, or server partition is an entity which provides a method of sharing the resources under its direct control with another partition, virtualizing those resources in the process. The following defines the Server VIO model: ◆ The server is a server to a client. An example of this is the VSCSI client (see Section 16.5, "Virtual SCSI (VSCSI)," on page 575). In this case, the Server VIO model is one where the server gets access to the client partition's local memory via what the client mapped into an RTCE table. This access is done through the second window pane of the server's **"ibm,my-dma-window"** property, which is linked to the first window pane of the client's **"ibm,my-dma-window"** property.
Partner partition	This is "the other" partition in a pair of partitions which are connected via a virtual IOA pair. For client partitions, the partner is generally the server (although, in limited cases, client to client connections may be possible). For server partitions, the partner can be a client partition or another server partition.
RTCE table	Remote DMA TCE table. TCE (Translation Control Entry) and RTCE tables are used to translate I/O DMA operations and provide protection against improper operations (access to what should not be accessed or for protection against improper access modes, like writing to a read only page). More information on TCEs and TCE tables, which are used for physical IOAs, can be found in Section 3.2.2.2, "DMA Address Translation and Control via the TCE Mechanism," on page 65. The RTCE table for Remote DMA (RDMA) is analogous to the TCE table for physical IOAs. The RTCE table does, however, have a little more information in it (as placed there by the hypervisor) in order to, among other things, allow the hypervisor to create links to physical IOA TCEs that were created from the RTCE table TCEs. A TCE in an RTCE table is never accessed directly by the partitions software; only though hypervisor hcall()s. For more information on RTCE table and operations, see Section 17.2.1.2, "RTCE Table and Properties of the Children of the /vdevice Node," on page 601, and Section 17.2.3, "Partition Managed Class - Synchronous Infrastructure," on page 637.
Window pane (**"ibm,my-dma-window"** property)	The RTCE tables for VIO DMA are pointed to by the **"ibm,my-dma-window"** property in the device tree for each virtual device. This property can have one, two, or three triples, each consisting of a Logical I/O Bus Number (LIOBN), phys which is 0, and size. The LIOBN essentially points to a unique RTCE table (or a unique entry point into a single table. The phys is a value of 0, indicating offsets start at 0. The size is the size of the available address space for mapping memory into the RTCE table. This architecture talks about these unique RTCE tables as being window panes within the **"ibm,my-dma-window"** property. Thus, there can be up to three window panes for each virtual IOA, depending on the type of IOA. For more on usage of the window panes, see Table 226, "VIO Window Pane Usage and Applicable Hcall()s," on page 602.
RDMA	Remote Direct Memory Access is DMA transfer from the server to its client or from the server to its partner partition. DMA refers to both physical I/O to/from memory operations and to memory to memory move operations.
Copy RDMA	This term refers to when the hypervisor is used (possibly with hardware assist) to move data between server partition and client partition memories or between server partition and partner partition memories. See Section 17.2.3.2.1, "Copy RDMA," on page 642.
Redirected RDMA	This term refers to when the TCE(s) for a physical IOA are set up through the use of the RTCE table manipulation hcall()s (for example, H_PUT_RTCE) such that the client or partner's partition's RTCE table (though the second window pane of the server partition) is used by the hypervisor during the processing of the hcall() to setup the TCE(s) for the physical IOA, and then the physical IOA DMAs directly to or from the client or partner partition's memory. See Section 17.2.2.2, "Redirected RDMA (Using H_PUT_RTCE, and H_PUT_RTCE_INDIRECT)," on page 625 for more information.
LRDMA	Stands for Logical Remote DMA and refers to the set of facilities for synchronous RDMA operations. See also Section 17.2.3.2, "Logical Remote DMA (LRDMA) Option," on page 642 for more information. LRDMA is a separate option.

Table 224. Terminology used with VIO *(Continued)*

Term	Definition
Command/Response Queue (CRQ)	The CRQ is a facility which is used to communicate between partner partitions. Transport events which are signaled from the hypervisor to the partition are also reported in this queue.
Subordinate CRQ (Sub-CRQ)	Similar to the CRQ, except with notable differences (See Table 234, “CRQ and Sub-CRQ Comparison,” on page 634).
Reliable Command/Response Transport	This is the CRQ facility used for synchronous VIO operations to communicate between partner partitions. Several hcall()s are defined which allow a partition to place an entry on the partner partition’s queue. The firmware can also place transport change of status messages into the queue to notify a partition when the connection has been lost (for example, due to the other partition crashing or deregistering its queue). See Section 17.2.3.1, “Reliable Command/Response Transport Option,” on page 637 for more information.
Subordinate CRQ Transport	This is the Sub-CRQ facility used for synchronous VIO operations to communicate between partner partitions when the CRQ facility by itself is not sufficient. The Subordinate CRQ Transport never exists without a corresponding Reliable Command/Response Transport. See Section 17.2.3.3, “Subordinate CRQ Transport Option,” on page 645 for more information.

17.2 VIO Architectural Infrastructure

VIO is used in conjunction with the Logical Partitioning option as described in Chapter 14, “Logical Partitioning Option,” on page 385. For each of a platform’s partitions, the number and type of VIO adapters with the associated inter-partition communications paths (if any are defined). These definitions take the architectural form of VIO adapters and are communicated to the partitions as device nodes in their OF device tree. Depending upon the specific virtual device, their device tree node may be found as a child of `/` (the root node) or in the VIO sub-tree (see below).

The VIO infrastructure provides several primitives that may be used to build connections between partitions for various purposes (that is, for various virtual IOA types). These primitives include:

- A Command/Response Queue (CRQ) facility which provides a pipe between partitions. A partition can enqueue an entry on its partner’s CRQ for processing by that partner. The partition can set up the CRQ to receive an interrupt when the queue goes from empty to non-empty, and hence this facility provides a method for an inter-partition interrupt.
- A Subordinate CRQ (Sub-CRQ) facility that may be used in conjunction with the CRQ facility, when the CRQ facility by itself is not sufficient. That is, when more than one queue with more than one interrupt is required by the virtual IOA.
- An extended TCE table called the RTCE table which allows a partition to provide “windows” into the memory of its partition to its partner partition, while maintaining addressing and access control to its memory.
- Remote DMA services that allow a server partition to transfer data to a partner partition’s memory via the RTCE table window panes. This allows a device driver in a server partition to efficiently transfer data to and from a partner, which is key in sharing of an IOA in the server partition with its partner partition.

In addition to the virtual IOAs themselves, this architecture defines a virtual host bridge, and a virtual interrupt source controller. The virtual host bridge roots the VIO sub-tree. The virtual interrupt source controller provides the consistent syntax for communicating the interrupt numbers the partition’s OS sees when the virtual IOAs signal an interrupt.

The general VIO infrastructure is defined in Section 17.2.1, “VIO Infrastructure - General,” on page 600. There are additional infrastructures requirements for the partition managed class based on the Synchronous VIO model. See Section 17.2.3, “Partition Managed Class - Synchronous Infrastructure,” on page 637.

17.2.1 VIO Infrastructure - General

This section describes the general OF device tree structure for virtual IOAs and describes in more detail window panes, as well as describing the interrupt control aspects of virtual IOAs.

17.2.1.1 Properties of the `/vdevice` OF Tree Node

Most VIO adapters are represented in the OF device tree as children of the **`/vdevice`** node (child of the root node). While the vdevice sub-tree is the preferred architectural home for VIO adapters, selected devices for historical reasons, are housed outside of the vdevice sub-tree.

R1–17.2.1.1–1. The platform's **`/vdevice`** node must contain the properties as defined in Table 225, "Properties of the /vdevice Node," on page 600.

Table 225. Properties of the **`/vdevice`** Node

Property Name	Required?	Definition
"name"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the virtual device name, the value shall be "vdevice"
"device_type"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the virtual device type, the value shall be "vdevice"
"model"	NA	Property not present.
"compatible"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the virtual device programming models, the value shall include "IBM,vdevice"
"used-by-rtas"	NA	Property not present.
"ibm,loc-code"	NA	The location code is meaningless unless one is doing dynamic reconfiguration as in the children of this node.
"reg"	NA	Property not present.
"#size-cells"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], the value shall be 0. No child of this node takes space in the address map as seen by the owning partition.
"#address-cells"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], the value shall be 1.
"#interrupt-cells"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], the value shall be 2. The first cell contains the interrupt# as will appear in the XIRR and is used as input to interrupt RTAS calls. The second cell contains the value 0 indicating a positive edge sense
"interrupt-map-mask"	NA	Property not present.
"interrupt-ranges"	Y	Standard property name that defines the interrupt number(s) and range(s) handled by this unit.
"ranges"		These will probably be needed for IB virtual adapters.
"interrupt-map"	NA	Property not present.
"interrupt-controller"	Y	The **`/vdevice`** node appears to contain an interrupt controller.
"ibm,drc-indexes"	For DR	Refers to the DR slots -- the number provided is the maximum number of slots that can be configured which is limited by, among other things, the RTCE tables allocated by the hypervisor.

Table 225. Properties of the **/vdevice** Node

Property Name	Required?	Definition
"ibm,drc-power-domains"	For DR	Value of -1 to indicate that no power manipulation is possible or needed.
"ibm,drc-types"	For DR	Value of "SLOT". Any virtual IOA can fit into any virtual slot.
"ibm,drc-names"	For DR	The virtual location code (see Section 12.3.2.16, "Virtual Card Connector Location Codes," on page 335)
"ibm,max-virtual-dma-size"	See definition column	The maximum transfer size for H_SEND_LOGICAL_LAN and H_COPY_RDMA hcall()s. Applies to all VIO which are children of the **/vdevice** node. Minimum value is 128 KB.

17.2.1.2 RTCE Table and Properties of the Children of the /vdevice Node

This architecture defines an extended type of TCE table called a Remote DMA TCE (RTCE) table. An RTCE table is one that is not directly used by the hardware to translate an I/O adapter's DMA addresses, but is used by the hypervisor to translate a partition's I/O addresses. RTCE tables have extra data, compared with a standard TCE table, to help firmware manage the use of its mappings. A partition manages the entries for its memory that is to be the target for I/O operations in the RTCE table using the TCE manipulation hcall()s, depending on the type of window pane. More on this later in this section. On platforms implementing the CRQ LRDMA options, these hcall()s are extended to understand the format of the RTCE table via the LIOBN parameter that is used to address the specific window pane within an RTCE table[1].

Children of the **/vdevice** node that support operations which use RTCE tables (for example, RDMA) contain the **"ibm,my-dma-window"** property. This property contains one or more (*logical-I/O-bus-number, phys*, *size*) triple(s). Each triple represents one window pane in an RTCE table which is available to this virtual IOA. The phys value is 0, and hence the logical I/O bus number (LIOBN) points to a unique range of TCEs in the RTCE table which are assigned to this window pane (LIOBN), and hence the I/O address for that LIOBN begin at 0.

The LIOBN is an opaque handle which references a window pane within an RTCE table. Since this handle is opaque, its internal structure is not architected, but left to the implementation's discretion. However, it is the architectural intent that the LIOBN be an indirect reference to the RTCE table through a hypervisor table that contains management variables, allowing for movement of the RTCE table and table format specific access methods. The partition uses an I/O address as an offset relative to the beginning of the LIOBN, as part of any I/O request to that memory mapped by that RTCE table's TCEs. A server partition appends its version of the LIOBN for the partner partition's RTCE table that represents the partner partition's RTCE table which it received through the second entry in the **"ibm,my-dma-window"** property associated with server partition's virtual IOA's device tree node (for example, see Table 225, "Properties of the /vdevice Node," on page 600). The mapping between the LIOBN in the second pane of a server virtual IOA's **"ibm,my-dma-window"** property and the corresponding partner partition IOA's RTCE table is made when the CRQ successfully completes registration.

The window panes and the hcall()s that are applicable to those panes, are defined and used as indicated in Table 226, "VIO Window Pane Usage and Applicable Hcall()s," on page 602.

1.One could also think of each LIOBN pointing to a separate RTCE table, rather than window panes within an RTCE table.

Table 226. VIO Window Pane Usage and Applicable Hcall()s

Window Pane (Which Triple)	Hypervisor Simulated Class	Client VIO Model	Server VIO Model
First	◆ I/O address range which is available to map local partition memory for use by the hypervisor	◆ I/O address range which is available to map local partition memory to make it available to the hypervisor use (access to the CRQ and any Sub-CRQs). ◆ For clients which support RDMA operations from their partner partition to their local memory (for example, VSCSI), this I/O address range is available to map local partition memory to make it available to the server partition, and this pane gets mapped to the second window pane of the partner partition (client/server relationship).	◆ I/O address range which is available to map local partition memory for use by the hypervisor (for access by H_COPY_RDMA requests, and for access to the CRQ, any Sub-CRQs). ◆ This window is not available to any other partition.
	Applicable hcall()s: H_PUT_TCE, H_GET_TCE, H_PUT_TCE_INDIRECT, H_STUFF_TCE		
Second	Does not exist	Does not exist	◆ I/O address range which corresponds to a window pane of the partner partition: linked to the first window pane for Client/Server model connections. ◆ Used to get access to the partner partition's memory from the hypervisor that services the local partition for use as source or destination in Copy RDMA requests or for redirected DMA operations (for example, H_PUT_RTCE).
	Applicable hcall()s: H_PUT_RTCE, H_REMOVE_RTCE and H_PUT_RTCE_INDIRECT		

The **"ibm,my-dma-window"** property is the per device equivalent of the **"ibm,dma-window"** property found in nodes representing bus bridges.

Children of the **/vdevice** node contain virtual location codes in their **"ibm,loc-code"** properties. The invariant assignment number is uniquely generated when the virtual IOA is assigned to the partition and remains invariably associated with that virtual IOA for the duration of the partition definition. For more information, see Section 12.3.2.16, "Virtual Card Connector Location Codes," on page 335.

17.2.1.3 VIO Interrupt Control

There are two hcall()s that work in conjunction with the RTAS calls *ibm,int-on*, *ibm,int-off*, *ibm,set-xive* and *ibm,get-xive*, which manage the state of the interrupt presentation controller logic. These hcall()s provide the equivalent of IOA control registers used to control IOA interrupts. The usage of these two hcall()s is summarized in Table 227, "VIO Interrupt Control hcall() Usage," on page 602. The detail of the H_VIO_SIGNAL is shown after this table and the detail of the applicable H_VIOCTL subfunctions can be found in Section 17.2.1.6.10, "DISABLE_ALL_VIO_INTERRUPTS Subfunction Semantics," on page 618, Section 17.2.1.6.11, "DISABLE_VIO_INTERRUPT Subfunction Semantics," on page 618, and Section 17.2.1.6.12, "ENABLE_VIO_INTERRUPT Subfunction Semantics," on page 618.

Table 227. VIO Interrupt Control hcall() Usage

Interrupt From	Virtual IOA Definition does *not* Include Sub-CRQs	Virtual IOA Definition *Includes* Sub-CRQs	Interrupt Number Obtained From
CRQ	H_VIO_SIGNAL	H_VIO_SIGNAL or H_VIOCTL	OF device tree **"interrupts"** property
Sub-CRQ	Not Applicable	H_VIOCTL	H_REG_SUB_CRQ hcall()

17.2.1.3.1 H_VIO_SIGNAL

This H_VIO_SIGNAL hcall() manages the interrupt mode of a virtual adapter's CRQ interrupt signalling logic. There are two modes: Disabled, and Enabled.

The first interrupt of the **"interrupts"** property is for the CRQ.

Syntax:

```
int64                   /* H_Success: Expected return code */
                        /* H_Parameter: One or more of the input parameters are invalid *
                        /* H_Hardware: A hardware problem prevented completion of the operation*/
    hcall (const int64 H_VIO_SIGNAL,  /* Function Code */
            uint64 unit-address,      /* As specified in the Virtual IOA's device tree node */
            uint64 mode);             /* 0=Disabled, 1=Enabled with each bit representing a possible interrupt*/
```

Parameters:

- unit-address: unit address per device tree node **"reg"** property.
- mode:
  - Bit 63 controls the first interrupt specifier given in the virtual IOA's **"interrupts"** property, and bit 62 the second. High order bits not associated with an interrupt source as defined in the previous sentence should be set to zero by the caller and ignored by the hypervisor.
  - A bit value of 1 enables the specified interrupt, a bit value of 0 disables the specified interrupt.

Semantics:

- Validate that the unit address belongs to the partition and to a vdevice IOA, else H_Parameter.
- Validate that the mode is one of those defined, else H_Parameter.
- Establish the specified mode.
- Return H_Success.

17.2.1.4 General VIO Requirements

R1–17.2.1.4–1. For all VIO options: The platform must be running in LPAR mode.

R1–17.2.1.4–2. For all VIO options: The platform's OF device tree must include, as a child of the **root** node, a node of type **vdevice** as the parent of a sub-tree representing the virtual IOAs assigned to the partition (see Appendix B, "LoPAPR Binding," on page 661 for details).

R1–17.2.1.4–3. For all VIO options: The platform's **/vdevice** node must contain properties as defined in Table 225, "Properties of the /vdevice Node," on page 600.

R1–17.2.1.4–4. For all VIO options: If the platform is going to limit the size of virtual I/O data copy operations (e.g., H_SEND_LOGICAL_LAN and H_COPY_RDMA), then the platform's **/vdevice** node must contain the **"ibm,max-virtual-dma-size"** property, and the value of this property must be at least 128 KB.

R1–17.2.1.4–5. For all VIO options: The interrupt server numbers for all interrupt source numbers, virtual and physical, must come from the same name space and are defined by the **"ibm,interrupt-buid-size"** property in the **PowerPC External Interrupt Presentation Controller** Node.

R1–17.2.1.4–6. For all VIO options: The virtual interrupts for all children of the **`/vdevice`** node must, upon transfer of control to the booted partition program, be masked as would be the result of an *ibm,int-off* RTAS call specifying the virtual interrupt source number.

R1–17.2.1.4–7. For all VIO options with the Reliable Command/Response option: The platform must specify the CRQ interrupt as the first interrupt in the **`"interrupts"`** property for a virtual IOA.

R1–17.2.1.4–8.

R1–17.2.1.4–9. For all VIO options: The platform must implement the H_VIO_SIGNAL hcall() as defined in Section 17.2.1.3, "VIO Interrupt Control," on page 602.

R1–17.2.1.4–10. For all VIO options: The platform must assign an invariant virtual location code to each virtual IOA as described in Section 12.3.2.16, "Virtual Card Connector Location Codes," on page 335**.**

R1–17.2.1.4–11. *(Requirement Number Reserved For Compatibility)*

R1–17.2.1.4–12. For all VIO options: The *phys* of each **`"ibm,my-dma-window"`** property triple (window pane) must have a value of zero and the LIOBN must be unique.

Implementation Note: While the architectural definition of LIOBN would allow the definition of one logical I/O bus number (LIOBN) for all RTCE tables (IOBA ranges separating IOAs), such an implementation is not permitted for the VIO option, which requires a unique LIOBN (at least per partition preferably platform wide) for each virtual IOA window pane. Such designs allow the LIOBN handle to be used to validate access rights, and allows each subsequent I/O bus address range to start at zero, providing maximum accommodation for 32 bit OS's.

R1–17.2.1.4–13. For the VSCSI option: For the server partition, there must exist two triples (two window panes) in the **`"ibm,my-dma-window"`** property and the size field of the second triple (second window pane) of an **`"ibm,my-dma-window"`** property must be equal to the size field of the corresponding first triple (first window pane) of the associated partner partition's **`"ibm,my-dma-window"`** property.

R1–17.2.1.4–14.

Implementation Note: In order to meet Requirement R1–17.2.1.4–13, it may be necessary for implementations to assign an implementation dependent default size to all RTCE tables.

R1–17.2.1.4–15. For all VIO options: RTCE tables for virtual IOAs, as pointed to by the partitions' first window pane of the **`"ibm,my-dma-window"`** property, and the TCEs that they contain (as built by the TCE hcall()s) must be persistent across partner partition reboots and across partner partition deregister (free)/re-register operations, even when the partition which connects after one deregisters is a different partition, and must be available to have TCEs built in them by said partition, as long as that partition still owns the corresponding virtual IOA (an LRDR operation which removes the IOA will also remove the RTCE table).

R1–17.2.1.4–16. For all VIO options: The connection between the second window pane of the **`"ibm,my-dma-window"`** property for a partition and its corresponding window pane in the partner partition (first window pane) must be broken by the platform when either partition deregisters its CRQ or when either partition terminates, and the platform must invalidate any redirected TCEs copied from the said second window pane (for information on invalidation of TCEs, see Section 17.2.2.2.4, "Redirected RDMA TCE Recovery and In-Flight DMA," on page 631)**.**

R1–17.2.1.4–17. For all VIO options: The following window panes of the **`"ibm,my-dma-window"`** property, when they exist, must support the following specified hcall()s, when they are implemented:

a. For the first window pane: H_PUT_TCE, H_GET_TCE, H_PUT_TCE_INDIRECT, H_STUFF_TCE

b. For the second window pane: H_PUT_RTCE, H_REMOVE_RTCE, H_PUT_RTCE_INDIRECT

R1–17.2.1.4–18. For all VIO options: The platform must not prohibit the server and partner partition, or client and partner partition, from being the same partition, unless the user interface used to setup the virtual IOAs specifically disallows such configurations.

R1–17.2.1.4–19. For all VIO options: Any child node of the **`/vdevice`** node that is not defined by this architecture must contain the **`"used-by-rtas"`** property.

Implementation Notes:

1. Relative to Requirement R1–17.2.1.4–18, partner partitions being the same partition makes sense from a product development standpoint.
2. The *ibm,partner-control* RTAS call does not make sense if the partner partitions are the same partition.

R1–17.2.1.4–20. For all VIO options: The platform must implement the H_VIOCTL hcall() following the syntax of Section 17.2.1.6, “H_VIOCTL,” on page 613 and semantics specified by Table 229, “Semantics for H_VIOCTL subfunction parameter values,” on page 614.

17.2.1.5 Shared Logical Resources

The sharing of resources, within the boundaries of a single coherence domain, owned by a partition owning a server virtual IOA by its clients (those owning the associated client virtual IOAs) is controlled by the hcall()s described in this section. The owning partition retains control of and access to the resources and can ask for their return or indeed force it. Refer to Figure 34, “Shared Logical Resource State Transitions,” on page 606 for a graphic representation of the state transitions involved in sharing logical resources.

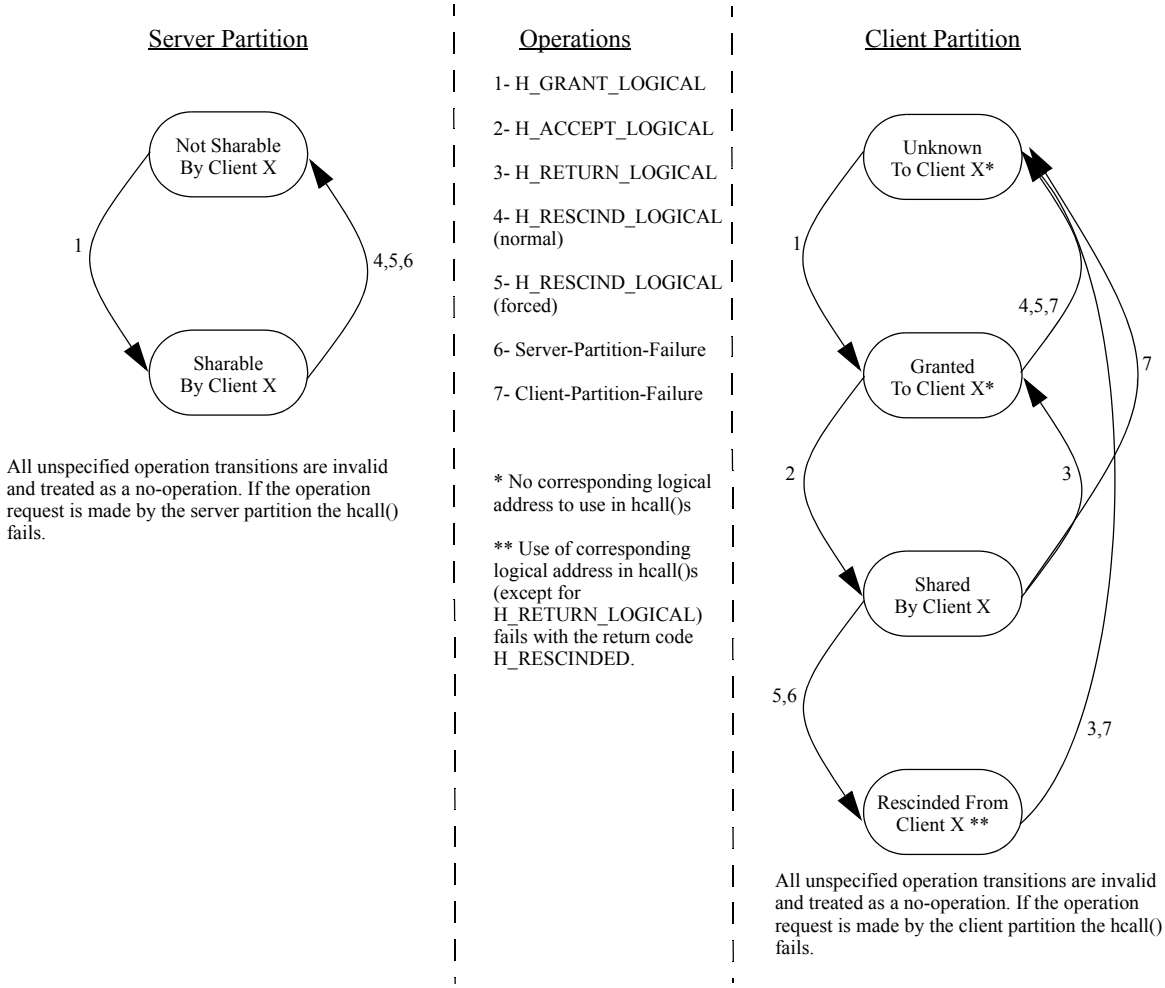


Figure 34. Shared Logical Resource State Transitions

Owners of resources can grant, to one or more client partitions, access to any of its resources. A client partition being defined as a partition with which the resource owner is authorized to register a CRQ, as denoted via an OF device tree virtual IOA node. Granting access is accomplished by requesting that the hypervisor generate a specific cookie for that resource for a specific sharing partition. The cookie value thus generated is unique only within the context of the partition being granted the resource and is unusable for gaining access to the resource by any other partition. This unique cookie is then communicated via some inter partition communications channel, most likely the authorized Command Response Queue. The partner partition then accepts the logical resource (mapping it into the accepting partition's logical address space). The owning partition may grant shared access of the same logical resource to several clients (by generating separate cookies for each client). During the time the resource is shared, both the owner and the sharer(s) have access to the logical resource, the software running in these partitions use private protocols to synchronize control access. Once the resource has been accepted into the client's logical address space, the resource can be used by the client in any way it wishes, including granting it to one of its own clients. When the client no longer needs access to the shared logical resource, it destroys any virtual mappings it may have created for the logical resource and returns the logical resource thus unmapping it from its logical address space. The client program could, subsequently accept the logical resource again (given that the cookie is still valid). To complete the termination of sharing, the owner partition rescinds the cookie describing the shared resource. Normally a rescind operation succeeds only if the client has re-

turned the resource, however, the owner can force the rescind in cases where it suspects that the client is incapable of gracefully returning the resource.

In the case of a forced rescind, the hypervisor marks the client partition's logical address map location corresponding to the shared logical resource such that any future hcall() that specifies the logical address fails with an H_RESCINDED return code. The hypervisor then ensures that the client partition's translation tables contain no references to a physical address of the shared logical resource.

Should the server partition fail, the hypervisor automatically notifies client partitions of the fact via the standard CRQ event message. In addition, the hypervisor recovers any outstanding shared logical resources prior to restarting the server partition. This recovery is proceeded by a minimum of two seconds of delay to allow the client partitions time to gracefully return the shared logical resources, then the hypervisor performs the equivalent of a forced rescind operation on all the server partition's outstanding shared logical resources.

This architecture does not specify a method of implementation, however, for the sake of clarifying the specified function, the following example implementation is given, refer to Figure 35, "Example Implementation of Control Structures for Shared Logical Resources," on page 608. Assume that the hypervisor maintains for each partition a logical to physical translation table (2) (used to verify the partition's virtual to logical mapping requests). Each logical resource (4) mapped within the logical to real translation table has associated with it a logical resource control structure (3) (some of the contents of this control structure are defined in the following text). The original logical resource control structures (3) describe the standard logical resources allocated to the partition due to the partition's definition, such as one per Logical Memory Blocks (LMB), etc.

The platform firmware, when creating the OF device tree for a given partition knows the specific configuration of virtual IOAs with the associated quantity of the various types of logical resources types for each virtual IOA. From that knowledge, it understands the number and type of resources that must be shared between the server and client partitions and therefore the number of control structures that will be needed. When an owning partition grants access to a subset of one of its logical resources to another partition, the hypervisor chooses a logical resource control structure to describe this newly granted resource (6), (as stated above, the required number of these control structures were allocated when the client virtual IOA was defined) and attaches it to the grantee's base partition control structure (5). This logical resource control structure is linked (9) to the base logical resource control structure (3) of the resource owner. Subsequently the grantee's OS may accept the shared logical resource (4) mapping it (7) into the grantee's partition logical to physical map table (8). This same set of operations may subsequently be performed for other partition(s) (10). The shared resource is always a subset (potentially the complete subset) of the original. Once a partition (10) has accepted a resource, it may subsequently grant a subset of that resource to yet another partition (14), here the hypervisor creates a logical resource control structure (13) links it (12) to the logical resource control structure (11) of the granting partition (10) that is in turn linked (9) to the owner's logical resource control structure (3).

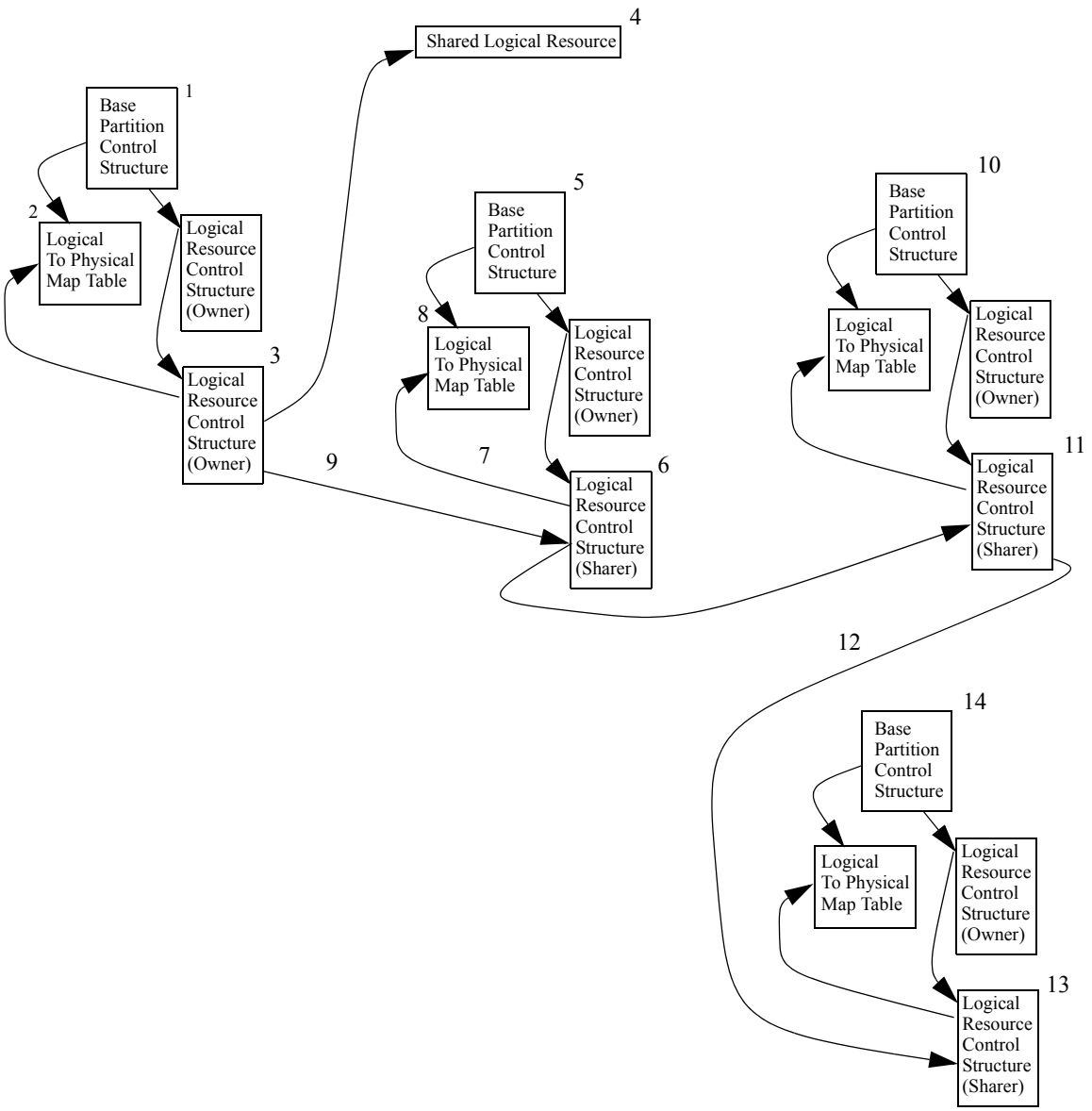


Figure 35. Example Implementation of Control Structures for Shared Logical Resources

For the OS to return the logical resource represented by control structure (11), the grant represented by control structure (13) needs to be rescinded. This is normally accomplished only after the OS that is running partition (14) performs a return operation, either because it has finished using the logical resource, or in response to a request (delivered by inter partition communications channel) from the owner. The exceptions are in the case that either partition terminates (the return operation is performed by the hypervisor) and a non-responsive client (when the granter performs a forced rescind). A return operation is much like a logical resource dynamic reconfiguration isolate operation, the hypervisor removes the logical resource from the partition's logical to physical map table, to prevent new virtual to physical mappings of the logical resource, then ensures that no virtual to physical mappings of the logical resource are outstanding (this can either be accomplished synchronously by checking map counts etc. or asynchronously prior to the completion of the rescind operation.

R1–17.2.1.5–1. For the Shared Logical Resource option: The platform must implement the hcall-logical-resource function set following the syntax and semantics of the included hcall(s) as specified in:
Section 17.2.1.5.1, “H_GRANT_LOGICAL,” on page 609,
Section 17.2.1.5.2, “H_RESCIND_LOGICAL,” on page 611,
Section 17.2.1.5.3, “H_ACCEPT_LOGICAL,” on page 612, and
Section 17.2.1.5.4, “H_RETURN_LOGICAL,” on page 612.

R1–17.2.1.5–2. For the Shared Logical Resource option: In the event that the partition owning a granted shared logical resource fails, the platform must wait for a minimum of 2 seconds after notifying the client partitions before recovering the shared resources via an automatic H_RESCIND_LOGICAL (forced) operation.

17.2.1.5.1 H_GRANT_LOGICAL

This hcall() creates a cookie that represents the specific instance of the shared object. That is, the specific subset of the original logical resource to be shared with the specific receiver partition. The owning partition makes this hcall() in preparation for the sharing of the logical resource subset with the receiver partition. The resulting cookie is only valid for the specified receiver partition.

The caller needs to understand the bounds of the logical resource being granted, such as for example, the logical address range of a given LMB. The generated cookie does not span multiple elemental logical resources (that is resources represented by their own Dynamic Reconfiguration Connector). If the owner wishes to share a range of resources that does span multiple elemental logical resources, then the owner uses a series of H_GRANT_LOGICAL calls to generate a set of cookies, one for each subset of each elemental logical resource to be shared.

The “logical” parameter identifies the starting “address” of the subset of the logical resource to be shared. The form of this “address” is resource dependent, and is given in Table 228, “Format of H_GRANT_LOGICAL parameters,” on page 610.

Syntax:

```
uint64          /*      H_Success       Expected return code */
                /*      H_Hardware      Operation failed because of hardware error*/
                /*      H_Parameter     One or more parameters were in error */
                /*      H_Permission    A grant restriction precludes the operation*/
                /*      H_RESCINDED:    A specified parameter refers to a rescinded shared logical resource*/
                /*      H_NOMEM         Operation failed due to lack of hypervisor resources */
    hcall (     const uint64 H_GRANT_LOGICAL, /* Returns in R4 a cookie representing a shared logical resource */
                uint64  flags,          /* Resource type: */
                                                /*Main store */
                                                /*MMIO space */
                                                /* Interrupt Sources */
                                                /* DMA window panes*/
                                                /* Inter Processor Interrupt ports */
                                        /* Access restriction control bits (No R/W re-grant) */
                uint64  logical-hi,     /* identifier (storage logical address, LIOBN) */
                uint64  logical-lo,     /* identifier (storage logical address, interrupt source #, IOBA) */
                uint64  length,         /* The number of fundamental logical elements -- units per resource type */
                uint64  unit-address);  /* As specified in the Virtual IOA’s device tree node */
```

Parameters:

Table 228. Format of H_GRANT_LOGICAL parameters

Flags subfunction code (bits 16-23) value:		
Access Restriction	Bits 16-19	The defined bits in this field have independent meanings, and may appear in combination with all other bits unless specifically disallowed. (an x in the binary field indicates that the bit can take on a value of 0 or 1)
	0b1xxx	Read access inhibited (The grantee may not read from or grant access to read from this logical resource.)
	0bx1xx	Write access inhibited (The grantee may not write to or grant access to write to this logical resource.)
	0bxx1x	Re-Grant rights inhibited (the grantee may not grant access to this logical resource to a subsequent client.)
	0bxxx1	Reserved calling software should set this bit to zero. Firmware returns H_Parameter if set.

Logical Resource	Supported Combinations	Bits 20-23	"address" description	"length" description
System Memory	0bxxx0	0x1	Logical Address (as would be used in H_ENTER) in logical-lo; logical-hi not used (should be = 0)	Bytes in units of 4 K on 4 K boundaries (low order 12 bits = 0)
MMIO Space	0bxxx0	0x2	Logical Address (as would be used in H_ENTER) in logical-lo; logical-hi not used (should be = 0)	Bytes in units of 4 K on 4 K boundaries (low order 12 bits = 0)
Interrupt Source	0b00x0	0x4	24 bit Interrupt # (as would be used in *ibm,get-xive*) in low order 3 bytes in logical-lo; logical-hi not used (should be = 0)	value=1 (the logical resource is one indivisible unit)
DMA Window Pane[1]	0b00x0	0x5	32 bit LIOBN in logical-hi; with a 64 bit IOBA logical-lo	Bytes of IOBA in units of 4 K on 4 K boundaries (low order 12 bits = 0)
Interprocessor Interrupt Port	0b00x0	0x6	Processor Number. (As from the processor's Unit ID) in logical-lo; logical-hi not used (should be = 0).	value=1 (the logical resource is one indivisible unit)

[1] Note: The DMA window only refers to physical DMA windows not virtual DMA windows. Virtual DMA windows can be directly created with a client virtual IOA definition and need not be shared with those of the server.

unit-address: The unit address of the virtual IOA associated with the shared logical resource, and thus the partner partition that is to share the logical resource.

Semantics:

- Verify that the flags parameter specifies a supported logical resource type, else return H_Parameter.
- Verify that the logical address validly identifies a logical resource of the type specified by the flags parameter and owned/shared by the calling partition, else return H_Parameter; unless:
  - The logical address's page number represents a page that has been rescinded by the owner, then return H_RESCINDED.
  - There exists a grant restriction on the logical resource, then return H_Permission.
- Verify that the length parameter is of valid form for the resource type specified by the flags parameter and that it represents a subset (up to the full size) of the logical resource specified by the logical address parameter, else return H_Parameter.
- Verify that the unit-address is for a virtual IOA owned by the calling partition, else return H_Parameter.
- If the partner partition's client virtual IOA has sufficient resources, generate hypervisor structures to represent, for hypervisor management purposes, including any grant restrictions, the specified shared logical resource, else return H_NOMEM.
- Generate a cookie associated with the hypervisor structures created in the previous step that the partner partition associated with the unit-address can use to reference said structures via the H_ACCEPT_LOGICAL and

H_RETURN_LOGICAL hcall()s and the calling partition can use to reference said structures via the H_RESCIND_LOGICAL hcall().

- Place the cookie generated in the previous step in R4 and return H_Success.

17.2.1.5.2 H_RESCIND_LOGICAL

This hcall() invalidates a logical sharing as created by H_GRANT_LOGICAL above. This operation may be subject to significant delays in certain circumstances. Callers may experience an extended series of H_PARTIAL returns prior to successful completion of this operation.

If the sharer of the logical resource has not successfully completed the H_RETURN_LOGICAL operation on the shared logical resource represented by the specified cookie, the H_RESCIND_LOGICAL hcall() fails with the H_Resource return code unless the “force” flag is specified. The use of the “force” flag increases the likelihood that a series of H_PARTIAL returns will be experienced prior to successful completion. The “force” flag also causes the hcall() to recursively rescind any and all cookies that represent subsequent sharing of the logical resource. That is, if the original client subsequently granted access to any or all of the logical resource to a client, those cookies and any other subsequent grants are also rescinded.

Syntax:

```
uint64      /*      H_Success       Expected return code */
            /*      H_Hardware      Operation failed because of hardware error*/
            /*      H_Parameter     One or more parameters were in error */
            /*      H_Resource      The operation failed because resource is in use by the sharer */
            /*      H_PARTIAL       Rescind in progress call later */
  hcall (   const uint64 H_RESCIND_LOGICAL,/* Invalidates a cookie representing a shared logical resource */
            uint64    flags,        /* force (ignore resource in use - remove sharer's access) */
            uint64    cookie);      /* cookie representing the shared resource */
```

Parameters:

flags: The flags subfunction code field (bits 16-23) two values are defined 0x00 “normal”, and 0x01 “forced”.

cookie: The handle returned by H_GRANT_LOGICAL representing the logical resource to be rescinded.

Semantics:

- Verify that the cookie parameter references an outstanding instance of a shared logical resource owned/accepted by the calling partition, else return H_Parameter.
- Verify that the flags parameter is one of the supported values, else return H_Paramter.
- If the “force” flag is specified [1], then: perform the functions of H_RETURN_LOGICAL (cookie) as if called by the client partition. Note this involves forced rescinding any cookies generated by the client partition that refer to the logical resource referenced by the original cookie being rescinded.
- If the client partition has the resource referenced by cookie is in the available for mapping via its logical to physical mapping table (the resource was accepted and not returned), return H_Resource.
- Verify that resource reference by cookie is not mapped by the client partition, else return H_PARTIAL.
- Hypervisor reclaims control structures referenced by cookie and returns H_Success.

1.Implementations should provide mechanisms to ensure that reserved flag field bits are zero, to improve performance, implementations may chose to activate this checking only in “debug” mode. The mechanism for activating an implementation dependent debug mode is outside of the scope of this architecture.

17.2.1.5.3 H_ACCEPT_LOGICAL

The H_ACCEPT_LOGICAL hcall() maps the granted logical resource into the client partition's logical address space. To provide the most compact client logical address space, the hypervisor maps the resource into the lowest applicable logical address for the referenced logical resource type, consistent with the resource's size and the resource type's constraints upon alignment etc. The chosen logical address for the starting point of the logical mapping is returned in register R4.

Syntax:

```
uint64      /*      H_Success       Expected return code */
            /*      H_Parameter     One or more parameters were in error */
            /*      H_RESCINDED:    A specified parameter refers to a rescinded shared logical resource/
            /*      H_Hardware      Operation failed because of hardware error*/
    hcall ( const uint64 H_ACCEPT_LOGICAL,/* Returns in R4 handle of a shared logical resource */
            uint64  cookie);        /* Cookie representing the shared resource */
```

Parameters:

cookie: The handle returned by H_GRANT_LOGICAL representing the logical resource to be accepted.

Semantics:

- Verify that the cookie parameter is valid for the calling partition, else return H_Parameter.
  - If the cookie represents a logical resources that has been rescinded by the owner, return H_RESCINDED.
- Map the resources represented by the cookie parameter, with any attendant access restrictions, into the lowest available logical address of the calling partition consistent with constraints of size and alignment and place the selected logical address into R4.
- Return H_Success.

17.2.1.5.4 H_RETURN_LOGICAL

The H_RETURN_LOGICAL hcall() unmaps the logical resource from the client partition's logical address space. Prior to calling H_RETURN_LOGICAL the client partition should have destroyed all virtual mappings to the section of the logical address space to which the logical resource is mapped. That is unmapping virtual addresses for MMIO and System Memory space, invalidating TCEs mapping a shared DMA window pane, disabling/masking shared interrupt sources and/or inter processor interrupts. Failing to do so, may result in parameter errors for other hcall()s and H_Resource from the H_RETURN_LOGICAL hcall(). Part of the semantic of this call is to determine that no such active mapping exists. Implementations may be able to determine this quickly if they for example maintain map counts for various logical resources, if an implementation searches a significant amount of platform tables, then the hcall() may return H_Busy and maintain internal state to continue the scan on subsequent calls using the same cookie parameter. The cookie parameter remains valid for the calling client partition until the server partition successfully executes the H_RESCIND_LOGICAL hcall().

Syntax:

```
uint64      /*      H_Success       Expected return code */
            /*      H_Busy          The operation is not yet complete call again */
            /*      H_Hardware      Operation failed because of hardware error*/
            /*      H_Parameter     One or more parameters were in error */
            /*      H_Resource      The operation failed because resource is in use by the sharer */
    hcall ( const uint64 H_RETURN_LOGICAL,/* Removes the logical resource from the partition's logical */
```

```
                                        /* address map*/
        uint64    cookie);              /* Cookie representing the shared resource */
```

Parameters:

cookie: The handle returned by H_GRANT_LOGICAL representing the logical resource to be returned.

Semantics:

- Verify that the cookie parameter references an outstanding instance of a shared logical resource accepted by the calling partition, else return H_Parameter.
- Remove the referenced logical resource from the calling partition's logical address map.
- Verify that no virtual to logical mappings exist for the referenced resource, else return H_Resource.
  - This operation may require extensive processing -- in some cases the hcall may return H_Busy to allow for improved system responsiveness -- in these cases the state of the mapping scan is retained in the hypervisor's state structures such that after some number of repeated calls the function is expected to finish.
- Return H_Success.

17.2.1.6 H_VIOCTL

The H_VIOCTL hypervisor call allows the partition to manipulate or query certain virtual IOA behaviors.

Syntax:

```
int64                                   /* H_Success,      Expected Return Code */
                                        /* H_Parameter,    One or more parameters were invalid */
                                        /* H_Constrained, The operation failed because of a resource constraint */
                                        /* H_Hardware,     The operation failed because of a hardware error */
                                        /* H_Not_Found,   Subfunction parameter value not supported *
                                        /* H_Closed,       The virtual I/O connection is closed */
        hcall(const unit64 H_VIOCTL,    /* Query/Set behaviors for the virtual IOA */
              uint64 unit-address,      /* As specified in the virtual IOA device tree node */
              uint64 subfunction,       /* The subfunction is encoded below */
              uint64 parm-1,            /* The last three parameters are defined in the semantics */
              uint64 parm-2,            /* for the specific subfunction. All unused parameters must */
              uint64 parm-3);           /* be set to zero */
```

Parameters:

- unit-address: As specified.
- subfunction: Specific subfunction to perform; see Table 229, "Semantics for H_VIOCTL subfunction parameter values," on page 614.
- parm-1: Specified in subfunction semantics.
- parm-2: Specified in subfunction semantics.
- parm-3: Specified in subfunction semantics.

Semantics:

- Validate that the subfunction is implemented, else return H_Not_Found.
- Validate the unit-address, else return H_Parameter.

- Validate the subfunction is valid for the given virtual IOA, else return H_Parameter.
- Refer to Table 229, “Semantics for H_VIOCTL subfunction parameter values,” on page 614 to determine the semantics for the given subfunction.

Table 229. Semantics for H_VIOCTL subfunction parameter values

Subfunction Number	Subfunction Name	Required?	Semantics Defined in
0x0	(Reserved)	(Reserved)	(Reserved)
0x1	GET_VIOA_DUMP_SIZE	For all VIO options	Section 17.2.1.6.1, “GET_VIOA_DUMP_SIZE Subfunction Semantics,” on page 615.
0x2	GET_VIOA_DUMP		Section 17.2.1.6.2, “GET_VIOA_DUMP Subfunction Semantics,” on page 615.
0x3	GET_ILLAN_NUMBER_VLAN_IDS	For the ILLAN option	Section 17.2.1.6.3, “GET_ILLAN_NUMBER_VLAN_IDS Subfunction Semantics,” on page 615.
0x4	GET_ILLAN_VLAN_ID_LIST		Section 17.2.1.6.4, “GET_ILLAN_VLAN_ID_LIST Subfunction Semantics,” on page 616.
0x5	GET_ILLAN_SWITCH_ID	For the ILLAN option	Section 17.2.1.6.5, “GET_ILLAN_SWITCH_ID Subfunction Semantics,” on page 616
0x6	DISABLE_MIGRATION	For all vscsi-server and vfc-server	Section 17.2.1.6.6, “DISABLE_MIGRATION Subfunction Semantics,” on page 616.
0x7	ENABLE_MIGRATION		Section 17.2.1.6.7, “ENABLE_MIGRATION Subfunction Semantics,” on page 616.
0x8	GET_PARTNER_INFO		Section 17.2.1.6.8, “GET_PARTNER_INFO Subfunction Semantics,” on page 617.
0x9	GET_PARTNER_WWPN_LIST	For all vfc-server	Section 17.2.1.6.9, “GET_PARTNER_WWPN_LIST Subfunction Semantics,” on page 617.
0xA	DISABLE_ALL_VIO_INTERRUPTS	For the Subordinate CRQ Transport option	Section 17.2.1.6.10, “DISABLE_ALL_VIO_INTERRUPTS Subfunction Semantics,” on page 618
0xB	DISABLE_VIO_INTERRUPT		Section 17.2.1.6.11, “DISABLE_VIO_INTERRUPT Subfunction Semantics,” on page 618
0xC	ENABLE_VIO_INTERRUPT		Section 17.2.1.6.12, “ENABLE_VIO_INTERRUPT Subfunction Semantics,” on page 618
0xD	GET_ILLAN_MAX_VLAN_PRIORITY	No	Section 17.2.1.6.13, “GET_ILLAN_MAX_VLAN_PRIORITY Subfunction Semantics,” on page 619
0xE	GET_ILLAN_NUMBER_MAC_ACLS	No	Section 17.2.1.6.14, “GET_ILLAN_NUMBER_MAC_ACLS Subfunction Semantics,” on page 619
0xF	GET_MAC_ACLS	No	Section 17.2.1.6.15, “GET_MAC_ACLS Subfunction Semantics,” on page 619
0x10	GET_PARTNER_UUID	For UUID Option	Section 17.2.1.6.16, “GET_PARTNER_UUID Subfunction Semantics,” on page 619
0x11	FW_RESET	For the VNIC option.	Section 17.2.1.6.17, “FW_Reset Subfunction Semantics,” on page 619

Table 229. Semantics for H_VIOCTL subfunction parameter values

Subfunction Number	Subfunction Name	Required?	Semantics Defined in
0x12	Get_ILLAN_SWITCHING_MODE	For any ILLAN adapter with the **"ibm,trunk-adapter"** property	
0x13	DISABLE_INACTIVE_TRUNK_RECEPTION	For any ILLAN adapter with the **"ibm,trunk-adapter"** property	Section 17.2.1.6.19, "DISABLE_INACTIVE_TRUNK_RECEPTION Subfunction Semantics:," on page 620

17.2.1.6.1 GET_VIOA_DUMP_SIZE Subfunction Semantics

- Validate parm-1, parm-2, and parm-3 are set to zero, else return H_Parameter.
- The hypervisor calculates the size necessary for passing opaque firmware data describing current virtual IOA state to the partition for purposes of error logging and RAS, and returns H_Success, with the required size in R4.

17.2.1.6.2 GET_VIOA_DUMP Subfunction Semantics

- If the given virtual IOA has an **"ibm,my-dma-window"** property in its device tree, then parm-1 is an eight byte output descriptor. The high order byte of an output descriptor is control, the next three bytes are a length field of the buffer in bytes, and the low order four bytes are a TCE mapped I/O address of the start of a buffer in I/O address space. The high order control byte must be set to zero. The TCE mapped I/O address is mapped via the first window pane of the **"ibm,my-dma-window"** property.
- If the given virtual IOA has no **"ibm,my-dma-window"** property in its device tree, then parm-1 shall be a logical real, page-aligned address of a 4 K page used to return the virtual IOA dump.
- Validate parm-2 and parm-3 are set to zero, else return H_Parameter.
- If parm-1 is an output descriptor, then
  - Validate the I/O address range is in the required DMA window and is mapped by valid TCEs, else return H_Parameter.
  - Transfer as much opaque hypervisor data as fits into the output buffer as specified by the output descriptor.
  - If all opaque data will not fit due to size, return H_Constrained, else return H_Success.
- If parm-1 is a logical real address, then
  - Validate the logical real address is valid for the partition, else return H_Parameter.
  - Transfer as much opaque hypervisor data as will fit into the passed logical real page, with a maximum of 4 K.
  - If all opaque data will not fit in the page due to size, return H_Constrained, else return H_Success.

17.2.1.6.3 GET_ILLAN_NUMBER_VLAN_IDS Subfunction Semantics

- Validate parm-1, parm-2, and parm-3 are set to zero, else return H_Parameter.
- The hypervisor returns H_Success, with the number of VLAN IDs (PVID + VIDs) in R4. This subfunction allows the partition to allocate the correct amount of space for the call: H_VIOCTL(GET_VLAN_ID_LIST).

17.2.1.6.4 GET_ILLAN_VLAN_ID_LIST Subfunction Semantics

- parm-1 is an eight byte output descriptor. The high order byte of an output descriptor is control, the next three bytes are a length field of the buffer in bytes, and the low order four bytes are a TCE mapped I/O address of the start of a buffer in I/O address space. The high order control byte must be set to zero. The TCE mapped I/O address is mapped via the first window pane of the **"ibm,my-dma-window"** property.
- Validate parm-2 and parm-3 are set to zero, else return H_Parameter.
- Validate the I/O address range is in the required DMA window and is mapped by valid TCEs, else return H_Parameter.
- Transfer the VLAN_ID_LIST into the output buffer as specified by the output descriptor. The data will be an array of two byte values, where the first element of the array is the PVID followed by all the VIDs. The format of the elements of the array is specified by IEEE VLAN documentation. Any unused space in the output buffer will be zeroed.
- If all VLAN IDs do not fit due to size, return H_Constrained.
- Return H_Success

17.2.1.6.5 GET_ILLAN_SWITCH_ID Subfunction Semantics

- parm-1 is an eight byte output descriptor. The high order byte of an output descriptor is control, the next three bytes are a length field of the buffer in bytes, and the low order four bytes are a TCE mapped I/O address of the start of a buffer in I/O address space. The high order control byte must be set to zero. The TCE mapped I/O address is mapped via the first window pane of the **"ibm,my-dma-window"** property.
- Validate parm-2 and parm-3 are set to zero, else return H_Parameter.
- Validate the I/O address range is in the required DMA window and is mapped by valid TCEs, else return H_Parameter.
- Transfer the GET_ILLAN_SWITCH_ID into the output buffer as specified by the output descriptor. The data will be a string of ASCII characters uniquely identifying the virtual switch to which the ILLAN adapter is connected. Any unused space in the output buffer will be zeroed.
- If the switch identifier does not fit due to size, return H_Constrained.
- Return H_Success

17.2.1.6.6 DISABLE_MIGRATION Subfunction Semantics

When this subfunction is implemented, the **"ibm,migration-control"** property exists in the **/vdevice** OF device tree node.

- Validate that parm-1, parm-2, and parm-3 are all set to zero, else return H_Parameter.
- If no partner is connected, then return H_Closed.
- Prevent the migration of the partner partition to the destination server until either the ENABLE_MIGRATION subfunction is called or H_FREE_CRQ is called.
- Return H_Success.

17.2.1.6.7 ENABLE_MIGRATION Subfunction Semantics

When this subfunction is implemented, the **"ibm,migration-control"** property exists in the **/vdevice** OF device tree node.

- Validate that parm-1, parm-2, and parm-3 are all set to zero, else return H_Parameter.
- Validate that the migration of the partner partition to the destination server was previously prevented with DISABLE_MIGRATION subfunction, else return H_Parameter.
- Enable the migration of the partner partition.
- Return H_Success.

17.2.1.6.8 GET_PARTNER_INFO Subfunction Semantics

- Parm-1 is an eight byte output descriptor. The high order byte of an output descriptor is control, the next three bytes are a length field of the buffer in bytes, and the low order four bytes are a TCE mapped I/O address of the start of a buffer in I/O address space. The high order control byte must be set to zero. The TCE mapped I/O address is mapped via the first window pane of the **"ibm,my-dma-window"** property.
- Validate parm-2 and parm-3 are set to zero, else return H_Parameter.
- Validate the I/O address range is in the required DMA window and is mapped by valid TCEs, else return H_Parameter.
- If the output buffer is not large enough to fit all the data, then return H_Constrained.
- If no partner is connected and more than one possible partner exists, then return H_Closed.
- Transfer the eight byte partner partition ID into the first eight bytes of the output buffer.
- Transfer the eight byte unit address into the second eight bytes of the output buffer.
- Transfer the NULL-terminated Converged Location Code associated with the partner unit address and partner partition ID immediately following the unit address.
- Zero any remaining output buffer.
- Return H_Success.

17.2.1.6.9 GET_PARTNER_WWPN_LIST Subfunction Semantics

This subfunction is used to get the WWPNs for the partner from the hypervisor. In this way, there is assurance that the WWPNs are accurate.

- Parm-1 is an eight byte output descriptor. The high order byte of an output descriptor is control, the next three bytes are a length field of the buffer in bytes, and the low order four bytes are a TCE mapped I/O address of the start of a buffer in I/O address space. The high order control byte must be set to zero. The TCE mapped I/O address is mapped via the first window pane of the **"ibm,my-dma-window"** property.
- Validate parm-2 and parm-3 are set to zero, else return H_Parameter.
- Validate the I/O address range is in the required DMA window and is mapped by valid TCEs, else return H_Parameter.
- If the output buffer is not large enough to fit all the data, return H_Constrained.
- If no partner is connected, return H_Closed.
- Transfer the first eight byte WWPN, which is represented in the **vfc-client** node of the partner partition in the **"ibm,port-wwn-1"** parameter, into the first eight bytes of the output buffer.
- Transfer the second eight byte WWPN, which is represented in the **vfc-client** node of the partner partition in the **"ibm,port-wwn-2"** parameter, into the second eight bytes of the output buffer.

- Zero any remaining output buffer.
- Return H_Success.

17.2.1.6.10 DISABLE_ALL_VIO_INTERRUPTS Subfunction Semantics

This subfunction is used to disable any and all the CRQ and Sub-CRQ interrupts associated with the virtual IOA designated by the unit-address, for VIOs that define the use of Sub-CRQs. Software that controls a virtual IOA that does not define the use of Sub-CRQ facilities should use the H_VIO_SIGNAL hcall() to disable CRQ interrupts.

Programming Note: On platforms that implement the partition migration option, after partition migration the support for this subfunction might change, and the caller should be prepared to receive an H_Not_Found return code indicating the platform does not implement this subfunction.

- Validate parm-1, parm-2, and parm-3 are set to zero, else return H_Parameter.
- Disable all CRQ and any Sub-CRQ interrupts associated with unit-address.
- Return H_Success.

17.2.1.6.11 DISABLE_VIO_INTERRUPT Subfunction Semantics

This subfunction is used to disable a CRQ or Sub-CRQ interrupt, for VIOs that define the use of Sub-CRQs. The CRQ or Sub-CRQ is defined by the unit-address and parm-1. Software that controls a virtual IOA that does not define the use of Sub-CRQ facilities should use the H_VIO_SIGNAL hcall() to disable CRQ interrupts.

Programming Note: On platforms that implement the partition migration option, after partition migration the support for this subfunction might change, and the caller should be prepared to receive an H_Not_Found return code indicating the platform does not implement this subfunction.

- Parm-1 is the interrupt number of the interrupt to be disabled. For an interrupt associated with a CRQ this number is obtained from the **"interrupts"** property in the device tree For an interrupt associated with a Sub-CRQ this number is obtained during the registration of the Sub-CRQ (H_REG_SUB_CRQ).
- Validate parm-1 is a valid interrupt number for a CRQ or Sub-CRQ for the virtual IOA defined by parm-1 and that parm-2 and parm-3 are set to zero, else return H_Parameter.
- Disable interrupt specified by parm-1.
- Return H_Success.

17.2.1.6.12 ENABLE_VIO_INTERRUPT Subfunction Semantics

This subfunction is used to enable a CRQ or Sub-CRQ interrupt, for VIOs that define the use of Sub-CRQs. The CRQ or Sub-CRQ is defined by the unit-address and parm-1. Software that controls a virtual IOA that does not define the use of Sub-CRQ facilities should use the H_VIO_SIGNAL hcall() to disable CRQ interrupts.

Programming Note: On platforms that implement the partition migration option, after partition migration the support for this subfunction might change, and the caller should be prepared to receive an H_Not_Found return code indicating the platform does not implement this subfunction.

- Parm-1 is the interrupt number of the interrupt to be enabled. For an interrupt associated with a CRQ this number is obtained from the **"interrupts"** property in the device tree For an interrupt associated with a Sub-CRQ this number is obtained during the registration of the Sub-CRQ (H_REG_SUB_CRQ).
- Validate parm-1 is a valid interrupt number for a CRQ or Sub-CRQ for the virtual IOA defined by unit-address and that parm-2 and parm-3 are set to zero, else return H_Parameter.
- Enable interrupt specified by parm-1.

- Return H_Success.

17.2.1.6.13 GET_ILLAN_MAX_VLAN_PRIORITY Subfunction Semantics

- Validate parm-1, parm-2, and parm-3 are set to zero, else return H_Parameter.
- The hypervisor returns H_Success, with the maximum IEEE 802.1Q VLAN priority returned in R4. If no priority limits are in place, the maximum VLAN priority is returned in R4.

17.2.1.6.14 GET_ILLAN_NUMBER_MAC_ACLS Subfunction Semantics

This subfunction allows the partition to allocate the correct amount of space for the GET_MAC_ACLS Subfunction call.

- Validate parm-1, parm-2, and parm-3 are set to zero, else return H_Parameter.
- The hypervisor returns H_Success, with the number of allowed MAC addresses returned in R4. If no MAC access control limits are in place, 0 is returned in R4.

17.2.1.6.15 GET_MAC_ACLS Subfunction Semantics

- parm-1 is an eight byte output descriptor. The high order byte of an output descriptor is control, the next three bytes are a length field of the buffer in bytes, and the low order four bytes are a TCE mapped I/O address of the start of a buffer in I/O address space. The high order control byte must be set to zero. The TCE mapped I/O address is mapped via the first window pane of the **"ibm,my-dma-window"** property.
- Validate parm-2 and parm-3 are set to zero, else return H_Parameter.
- Validate the I/O address range is in the required DMA window and is mapped by valid TCEs, else return H_Parameter.
- Transfer the allowed MAC addresses into the output buffer as specified by the output descriptor. The data will be an array of 8 byte values containing the allowed MAC address, with the low order 6 bytes containing the 6 byte MAC address. Any unused space in the output buffer are zeroed.
- If all allowed MAC addresses do not fit due to size, return H_Constrained.
- Return H_Success

17.2.1.6.16 GET_PARTNER_UUID Subfunction Semantics

- Validate parm-1, parm-2 and parm-3 are set to zero, else return H_Parameter.
- If no partner is connected and more than one possible partner exists, then return H_Closed.
- Transfer into registers R4 (High order 8 bytes) and R5 (low order 8 bytes) of the UUID of the client partition that owns the virtual device (See Section 7.3.16.21, "Universally Unique IDentifier," on page 233 for the format of the UUID string.
- Return H_Success

17.2.1.6.17 FW_Reset Subfunction Semantics

This H_VIOCTL subfunction will reset the VNIC firmware associated with a VNIC client adapter, if currently active. This subfunction is useful when the associated firmware becomes unresponsive to other CRQ-based commands. For the case of vTPMs the firmware will be left inoperable until the client partition next boots up.

Semantics:

- Validate that parm-1, parm-2, and parm-3 are all set to zero, else return H_Parameter.

- If the firmware associated with the virtual adapter can not be reset, return H_Constrained.
- Reset the firmware associated with the virtual adapter.
- Return H_Success.

17.2.1.6.18 GET_ILLAN_SWITCHING_MODE Subfunction Semantics

- Validate parm-1, parm-2, and parm-3 are set to zero, else return H_Parameter.
- Validate that the given virtual IOA is a ILLAN adapter with the "ibm,trunk-adapter", else return H_Parameter.
- The hypervisor returns H_Success, with the current switching mode in R4. If the switching mode is VEB mode, R4 will have a 0. If the switching mode is VEPA mode, R4 will have a 1.

17.2.1.6.19 DISABLE_INACTIVE_TRUNK_RECEPTION Subfunction Semantics:

This subfunction is used to disable the reception of all packets for a ILLAN trunk adapter that is not the Active Trunk Adapter as set by the H_ILLAN_ATTRIBUTES hcall.

Note: The default value for this attribute is ENABLED. The value is reset on a successful H_FREE_LOGICAL_LAN hcall or reboot/power change of the partition owning the ILLAN adapter.

- Validate parm-1, parm-2, and parm-3 are set to zero, else return H_Parameter.
- Validate that the given virtual IOA is a ILLAN adapter with the "ibm,trunk-adapter", else return H_Parameter.
- The hypervisor disables reception of packets for this adapter when it is not the Active Trunk Adapter.
- Return H_Success.

17.2.2 Partition Managed Class Infrastructure - General

In addition to the general requirements for all VIO described in Section 17.2.1.4, “General VIO Requirements,” on page 603, the architecture for the partition managed class of VIO defines several other infrastructures:

- A Command/Response Queue (CRQ) that allows communications back and forth between the server partition and its partner partition (see Section 17.2.2.1, “Command/Response Queue (CRQ),” on page 621).
- A Subordinate CRQ (Sub-CRQ) facility that may be used in conjunction with the CRQ facility, when the CRQ facility by itself is not sufficient. That is, when more than one queue with more than one interrupt is required by the virtual IOA. See Section 17.2.2.3, “Subordinate Command/Response Queue (Sub-CRQ),” on page 634.
- A mechanism for doing RDMA, which includes:
  - A mechanism called Copy RDMA that can be used by the device driver to move blocks of data between memory of the server and partner partitions
  - A mechanism for Redirected RDMA that allows the device driver to direct DMA of data from the server partition’s physical IOA to or from the partner partition’s memory (see Section 17.2.2.2, “Redirected RDMA (Using H_PUT_RTCE, and H_PUT_RTCE_INDIRECT),” on page 625).

The mechanisms for the synchronous type VIO are described as follows:

- Section 17.2.3, “Partition Managed Class - Synchronous Infrastructure,” on page 637

17.2.2.1 Command/Response Queue (CRQ)

The CRQ facility provides ordered delivery of messages between authorized partitions. The facility is reliable in the sense that the messages are delivered in sequence, that the sender of a message is notified if the transport facility is unable to deliver the message to the receiver's queue, and that a notification message is delivered (providing that there is free space on the receive queue), or if the partner partition either fails or deregisters its half of the transport connection. The CRQ facility does not police the contents of the payload portions (after the 1 byte header) of messages that are exchanged between the communicating pairs, however, this architecture does provide means (via the Format Byte) for self describing messages such that the definitions of the content and protocol between using pairs may evolve over time without change to the CRQ architecture, or its implementation.

The CRQ is used to hold received messages from the partner partition. The CRQ owner may optionally choose to be notified via an interrupt when a message is added to their queue.

17.2.2.1.1 CRQ Format and Registration

The CRQ is built of one or more 4 KB pages aligned on a 4 KB boundary within partition memory. The queue is organized as a circular buffer of 16 byte long elements. The queue is mapped into contiguous I/O addresses via the TCE mechanism and RTCE table (first window pane). The I/O address and length of the queue are registered by Section 17.2.3.1.1, "Reliable CRQ Format and Registration," on page 637. This registration process tells the hypervisor where to find the virtual IOA's CRQ.

17.2.2.1.2 CRQ Entry Format

Each CRQ entry consists of a 16 byte element. The first byte of a CRQ entry is the Header byte and is defined in Table 230, "CRQ Entry Header Byte Values," on page 621.

Table 230. CRQ Entry Header Byte Values

Header Value	Description
0	Element is unused -- all other bytes in the element are undefined
0x01 - 0x7F	Reserved
0x80	Valid Command/Response entry -- the second byte defines the entry format (for example, see Table 210, "Example Reliable CRQ Entry Format Byte Definitions for VSCSI," on page 577).
0x81 - 0xBF	Reserved
0xC0	Valid Initialization Command/Response entry -- the second byte defines the entry format. See Table 231, "Initialization Command/Response Entry Format Byte Definitions," on page 622.
0xC1 - 0xFE	Reserved
0xFF	Valid Transport Event -- the second byte defines the specific transport event. See Table 232, "Transport Event Codes," on page 622

The platform (transport mechanism) ignores the contents of all non-header bytes in all CRQ entries.

Valid Command/Response entries (Header byte 0x80) are used to carry data between communicating partners, transparently to the platform. The second byte of the entry is reserved for a Format byte to enable the definitions of the content and protocol between using pairs to evolve over time. The definition of the second byte of the Valid Command/Response entry is beyond the scope of this architecture. Table 210, "Example Reliable CRQ Entry Format Byte Definitions for VSCSI," on page 577 presents example VSCSI format byte values.

The Valid Initialization Command/Response entry (Header byte 0xC0) is used during virtual IOA initialization sequences. The second byte of this type entry is architected and is as defined in Table 231, “Initialization Command/Response Entry Format Byte Definitions,” on page 622. This format is used for initialization operations between communicating partitions. The remaining bytes (byte three and beyond) of the Valid Initialization Command/Response entry are available for definition by the communicating entities.

Table 231. Initialization Command/Response Entry Format Byte Definitions

Format Byte Value	Definition
0x0	Unused
0x1	Initialize
0x2	Initialization Complete
0x03 - 0xFE	Reserved
0xFF	Reserved for Expansion

Valid Transport Events (Header byte 0xFF) are used by the platform to notify communicating partners of conditions associated with the transport channel, such as the failure of the partner’s partition or the deregistration of the partner’s queue. The partner’s queue may be deregistered as a means of resetting the transport channel or simply to terminate the connection. When the Header byte of the queue entry specifies a Valid Transport Event, then the second byte of the CRQ entry defines the type of transport event. The Format byte (second byte) of a Valid Transport Event queue entry is architected and is as defined in Table 232, “Transport Event Codes,” on page 622).

Table 232. Transport Event Codes

Code Value	Explanation
0	Unused
1	Partner partition failed
2	Partner partition deregistered CRQ
3	
4	
5	
6	Partner partition suspended (for the Partition Suspension option)
0x07 - 0xFF	Reserved

The “partner partition suspended” transport event disables the associated CRQ such that any H_SEND_CRQ hcall() (See Section 17.2.3.1.5.3, “H_SEND_CRQ,” on page 640) to the associated CRQ returns H_Closed until the CRQ has been explicitly enabled using the H_ENABLE_CRQ hcall (See Section 17.2.3.1.5.4, “H_ENABLE_CRQ,” on page 641).

17.2.2.1.3 CRQ Entry Processing

Prior to the partition software registering the CRQ, the partition software sets all the header bytes to zero (entry invalid). After registration, the first valid entry is placed in the first element and the process proceeds to the end of the

queue and then wraps around to the first entry again (given that the entry has been subsequently marked as invalid). This allows both the partition software and transport firmware to maintain independent pointers to the next element they will be respectively using.

A sender uses an infrastructure dependent method to enter a 16 byte message on its partner's queue (see Section 17.2.3.1.3, "Reliable CRQ Entry Processing," on page 638). Prior to enqueueing an entry on the CRQ, the platform first checks if the session to the partner's queue is open, and there is a free entry, if not, it returns an error. If the checks succeed, the contents of the message is copied into the next free queue element, potentially notifying the receiver, and returns a successful status to the caller.

At the receiver's option, it may be notified via an interrupt when an element is enqueued to its CRQ. See "CRQ Facility Interrupt Notification" on page 623.

When the receiver has finished processing a queue entry, it writes the header to the value 0x00 to invalidate the entry and free it for future entries.

Should the receiver wish to terminate or reset the communication channel it deregisters the queue, and if it needs to re-establish communications, proceeds to register either the same or different section of memory as the new queue, with the queue pointers reset to the first entry.

17.2.2.1.4 CRQ Facility Interrupt Notification

The receiver can set the virtual interrupt associated with its CRQ to one of two modes. These are:

1. Disabled (An enqueue interrupt is not signaled.)
2. Enabled (An enqueue interrupt is signaled on every enqueue)

Note: An enqueue is considered a pulse not a level. The pulse then sets the memory element within the emulated interrupt source controller. This allows the resetting of the interrupt condition by simply issuing the H_EOI hcall() as is done with the PCI MSI architecture rather than having to do an explicit interrupt reset as in the case with PCI Level Sensitive Interrupt (LSI) architecture.

The interrupt mechanism is capable of presenting only one interrupt signal at a time from any given interrupt source. Therefore, no additional interrupts from a given source are ever signaled until the previous interrupt has been processed through to the issuance of an H_EOI hcall(). Specifically, even if the interrupt mode is enabled, the effect is to interrupt on an empty to non-empty transition of the queue. However, as with any asynchronous posting operation race conditions are to be expected. That is, an enqueue can happen in a window around the H_EOI hcall(). Therefore, the receiver should poll the CRQ after an H_EOI to prevent losing initiative.

See Section 17.2.1.3, "VIO Interrupt Control," on page 602) for information about interrupt control.

17.2.2.1.5 Extensions to Other hcall()s for CRQ

17.2.2.1.5.1 H_MIGRATE_DMA

Since the CRQ is RTCE table mapped, the H_MIGRATE_DMA hcall() may be requested to move a page that is part of the CRQ. The OS owner of the queue is responsible for preventing its processors from modifying the page during the migrate operation (as is standard practice with this hcall()), however, the H_MIGRATE_DMA hcall() serializes with the CRQ hcall()s to direct new elements to the migrated target page.

17.2.2.1.5.2 H_XIRR, H_EOI

The CRQ facility utilizes a virtual interrupt source number to notify the queue owner of new element enqueues. The standard H_XIRR and H_EOI hcall()s are extended to support this virtual interrupt mechanism, emulating the standard PowerPC Interrupt hardware with respect to the virtual interrupt source number.

17.2.2.1.6 CRQ Facility Requirements

R1–17.2.2.1.6–1. For the CRQ facility: The platform must implement the CRQ as specified in Section 17.2.2.1, “Command/Response Queue (CRQ),” on page 621.

R1–17.2.2.1.6–2. For the CRQ facility: The platform must reject CRQ definitions that are not 4 KB aligned.

R1–17.2.2.1.6–3. For the CRQ facility: The platform must reject CRQ definitions that are not a multiple of 4 KB long.

R1–17.2.2.1.6–4. For the CRQ facility: The platform must reject CRQ definitions that are not mapped relative to the TCE mapping defined by the first window pane of the virtual IOA’s **`“ibm,my-dma-window”`** property.

R1–17.2.2.1.6–5. For the CRQ facility: The platform must start enqueueing Commands/Responses to the newly registered CRQ starting at offset zero and proceeding as in a circular buffer, each entry being 16 byte aligned.

R1–17.2.2.1.6–6. For the CRQ facility: The platform must enqueue Commands/Responses only if the 16 byte entry is free (header byte contains 0x00), else the enqueue operation fails.

R1–17.2.2.1.6–7. For the CRQ facility: The platform must enqueue the 16 bytes specified in the validated enqueue request as specified in Requirement R1–17.2.2.1.6–5 except as required by Requirement R1–17.2.2.1.6–6.

R1–17.2.2.1.6–8. For the CRQ facility: The platform must not enqueue commands/response entries if the CRQ has not been registered successfully or if after a successful completion, has subsequently deregistered the CRQ.

R1–17.2.2.1.6–9. For the CRQ facility: The platform (transport mechanism) must ignore and must not modify the contents of all non-header bytes in all CRQ entries.

R1–17.2.2.1.6–10. For the CRQ facility: The first byte of a CRQ entry must be the Header byte and must be as defined in Table 230, “CRQ Entry Header Byte Values,” on page 621.

R1–17.2.2.1.6–11. For the CRQ facility: The Format byte (second byte) of a Valid Initialization CRQ entry must be as defined in Table 231, “Initialization Command/Response Entry Format Byte Definitions,” on page 622.

R1–17.2.2.1.6–12. For the CRQ facility: The Format byte (second byte) of a Valid Transport Event queue entry must be as defined in Table 232, “Transport Event Codes,” on page 622.

R1–17.2.2.1.6–13. For the CRQ facility: If the partner partition fails, then the platform must enqueue a 16 byte entry starting with 0xFF01 (last 14 bytes unspecified) as specified in Requirement R1–17.2.2.1.6–5 except as required by Requirements R1–17.2.2.1.6–6 and R1–17.2.2.1.6–8.

R1–17.2.2.1.6–14. For the CRQ facility: If the partner partition deregisters its corresponding CRQ, then the platform must enqueue a 16 byte entry starting with 0xFF02 (last 14 bytes unspecified) as specified in Requirement R1–17.2.2.1.6–5 except as required by Requirements R1–17.2.2.1.6–6 and R1–17.2.2.1.6–8.

R1–17.2.2.1.6–15.

R1–17.2.2.1.6–16. For the CRQ facility with the Partner Control option: If the partner partition is terminated by request of this partition via the *ibm,partner-control* RTAS call, then the platform must enqueue a 16 byte entry starting with 0xFF04 (last 14 bytes unspecified) as specified in Requirement R1–17.2.2.1.6–5 except as required by Requirements R1–17.2.2.1.6–6 and R1–17.2.2.1.6–8 when the partner partition has been successfully terminated.

R1–17.2.2.1.6–17.

R1–17.2.2.1.6–18. For the CRQ facility option: Platforms that implement the H_MIGRATE_DMA hcall() must implement that function for pages mapped for use by the CRQ.

R1–17.2.2.1.6–19. For the CRQ facility: The platforms must emulate the standard PowerPC External Interrupt Architecture for the interrupt source numbers associated with the virtual devices via the standard RTAS and hypervisor interrupt calls and must extend H_XIRR and H_EOI hcall()s as appropriate for CRQ interrupts.

R1–17.2.2.1.6–20. For the CRQ facility: The platform's OF must disable interrupts from the using virtual IOA before initially passing control to the booted partition program.

R1–17.2.2.1.6–21. For the CRQ facility: The platform's OF must disable interrupts from the using virtual IOA upon registering the IOA's CRQ.

R1–17.2.2.1.6–22. For the CRQ facility: The platform's OF must disable interrupts from the using virtual IOA upon deregistering the IOA's CRQ.

R1–17.2.2.1.6–23. For the CRQ facility: The platform must present (as appropriate per RTAS control of the interrupt source number) the partition owning a CRQ the appearance of an interrupt, from the interrupt source number associated, through the OF device tree node, with the virtual device, when a new entry is enqueued to the virtual device's CRQ and when the last interrupt mode set was "Enabled", unless a previous interrupt from the interrupt source number is still outstanding.

R1–17.2.2.1.6–24. For the CRQ facility: The platform must not present the partition owning a CRQ the appearance of an interrupt, from the interrupt source number associated, through the OF device tree node, with the virtual device, if the last interrupt mode set was "Disabled", unless a previous interrupt from the interrupt source number is still outstanding.

17.2.2.2 Redirected RDMA (Using H_PUT_RTCE, and H_PUT_RTCE_INDIRECT)

A server partition uses the hypervisor function, H_PUT_RTCE, which takes as a parameter the opaque handle (LIOBN) of the partner partition's RTCE table (second window pane of **`"ibm,my-dma-window"`**), an offset in the RTCE table, the handle for one of the server partition's I/O adapter TCE tables plus an offset within the I/O adapter's TCE table. H_PUT_RTCE then copies the appropriate contents of the partner partition's RTCE table into the server partition's I/O adapter TCE table. In effect, this hcall() allows the server partition's I/O adapter to have access to a specific section of the partner partition's memory as if it were the server partition's memory. However, the partner partition, through the hypervisor, maintains control over exactly which areas of the partner partition's memory are made available to the server partition without the overhead of the hypervisor having to directly handle each byte of the shared data.

The H_PUT_RTCE_INDIRECT, if implemented, takes as an input parameter a pointer to a list of offsets into the RTCE table, and builds the TCEs similar to the H_PUT_RTCE, described above.

A server partition uses the hypervisor function, H_REMOVE_RTCE, to back-out TCEs generated by the H_PUT_RTCE and H_PUT_RTCE_INDIRECT hcall()s.

The following rules guide the definition of the RTCE table entries and implementation of the H_PUT_RTCE, H_PUT_RTCE_INDIRECT, H_REMOVE_RTCE, H_MASS_MAP_TCE, H_PUT_TCE, H_PUT_TCE_INDIRECT, and H_STUFF_TCE hcall()s. Other implementations that provide the same external appearance as these rules are acceptable. The architectural intent is to provide RDMA performance essentially equivalent to direct TCE operations.

1. The partner partition's RTCE table is itself never directly accessed by an I/O Adapter (IOA), it is only accessed by the hypervisor, and therefore it can be a bigger structure than the regular TCE table as accessed by hardware (more fields).

2. When a server partition asks (via an H_PUT_RTCE or H_PUT_RTCE_INDIRECT hcall()) to have an RTCE table TCE copied to one of the server partition's physical IOA's TCEs, or asks (via an H_REMOVE_RTCE) to have an RTCE table entry removed from one of the server partition's physical IOA's TCEs, the hypervisor atomically, with respect to all RTCE table readers, sets (H_PUT_RTCE or H_PUT_RTCE_INDIRECT) or removes (H_REMOVE_RTCE) a field in the copied RTCE table entry[1]. This field is a pointer to the copy of the RTCE ta-

ble TCE in the server partition's IOA's TCE table. (A per RTCE table TCE lock is one method for the atomic setting of the copied RTCE table TCE link pointer.)

3. If a server partition tries to get another copy of the same RTCE table TCE it gets an error return (multiple mappings of the same physical page are allowed, they just need to use different RTCE table TCEs just like with physical IOA TCEs).

4. When the partner partition issues an H_PUT_TCE, H_PUT_TCE_INDIRECT, H_STUFF_TCE, or H_MASS_MAP hcall() to change his RTCE table, the hypervisor finds the TCE tables in one of several states. A number of these states represent unusual conditions, that can arise from timing windows or error conditions. The hypervisor rules for handling these cases are chosen to minimize its overhead while preventing one partition's errors from corrupting another partition's state.

   a. The RTCE table TCE is not currently in use: Clear/invalidate the TCE copy pointer and enter the RTCE table TCE mapping per the input parameters to the hcall().

   b. The RTCE table TCE contains a valid mapping and the TCE copy pointer is invalid (NULL or other implementation dependent value) (The previous mapping was never used for Redirected RDMA): Enter the RTCE table TCE mapping per the input parameters to the hcall().

   c. The RTCE table TCE contains a valid mapping and the TCE copy pointer references a TCE that does not contain a valid copy of the previous mapping in the RTCE table TCE. (The previous mapping was used for Redirected RDMA, however, the server partition has moved on and is no longer targeting the page represented by the old RTCE table TCE mapping): Clear/invalidate the TCE copy pointer and enter the RTCE table TCE mapping per the input parameters to the hcall().

   d. The RTCE table TCE contains a valid mapping and the TCE copy pointer references a TCE that does contain a valid copy of the previous mapping in the RTCE table TCE (the previous mapping is still potentially in use for Redirected RDMA, however, the partner partition has moved on and is no longer interested in the previous I/O operation). The server partition's IOA may still target a DMA operation against the TCE containing the copy of the RTCE table TCE mapping. The assumption is that any such targeting is the result of a timing window in the recovery of resources in the face of errors. Therefore, the server partition's TCE is considered invalid, but the server partition may or may not be able to immediately invalidate the TCEs. For more information on invalidation of TCEs, see Section 17.2.2.2.4, "Redirected RDMA TCE Recovery and In-Flight DMA," on page 631. The H_Resource return from an H_PUT_TCE, H_PUT_TCE_INDIRECT, and H_STUFF_TCE may be used to hold off invalidation in this case.

5. If a server partition terminates, the partner partition's device drivers time out the operations and resource recovery code recovers the RTCE table TCEs. If the partner partition terminates, the hypervisor scans the RTCE table and eventually invalidates all active copies of RTCE table TCEs. For more information on invalidation of TCEs, see Section 17.2.2.2.4, "Redirected RDMA TCE Recovery and In-Flight DMA," on page 631.

6. The server partition may use any of the supported hcall()s (see Section 17.2.1.2, "RTCE Table and Properties of the Children of the /vdevice Node," on page 601) to manage the TCE tables used by its IOAs. No extra restrictions are made to changes of the server partition's TCE table beside those stated in 2 above. The server partition can only target its own memory or the explicitly granted partner partition's memory.

7. The H_MIGRATE_DMA hcall() made by a partner partition migrates the page referenced by the RTCE table TCE but follows the RTCE table TCE copy pointer, if valid, to the server partition's IOA's TCE table to determine which IOA's DMA to disable, thus allowing migration of partner partition pages underneath server partition DMA activity. In this case, however, the H_MIGRATE_DMA algorithm is modified such that server partition's IOA's

1.This is an example of where the earlier statement "Other implementations that provide the same external appearance as these rules are acceptable" comes into affect. For example, for an RTCE table that is mapped with H_MASS_MAP_TCE, the pointer may not be in a field of the actual TCE in the RTCE table, but could, for example, be in a linked list, or other such structure, due to the fact that there is not a one-to-one correspondence from the RTCE to the physical IOA TCE in that case (H_MASS_MAP_TCE can map up to an LMB into one TCE, and physical IOA TCEs only map 4 KB).

TCE table is atomically updated, after the page migration but prior to enabling the IOA's DMA, only when its contents still are a valid copy of the partner partition's RTCE table TCE contents. The H_MIGRATE_DMA hcall() also serializes with H_PUT_RTCE so that new copies of the RTCE table TCE are not made during the migration of the underlying page.

8. The server partition should never call H_MIGRATE_DMA for any Redirected RDMA mapped pages, however, to check, the H_MIGRATE_DMA hcall() is enhanced to check the Logical Memory Block (LMB) owner in the TCE and reject the call if the LMB did not belong to the requester.

17.2.2.2.1 H_PUT_RTCE

This hcall() maps "count" number of contiguous TCEs in an RTCE to the same number of contiguous IOA TCEs. The H_REMOVE_RTCE hcall() is used to back-out TCEs built with H_PUT_RTCE hcall(). See Section 17.2.2.2.3, "H_REMOVE_RTCE," on page 630 for that hcall().

Syntax:

```
int64 hcall(    /* H_Success,              Expected Return Code
                H_Parameter,               One or more of the parameters are invalid -- no mappings changed
                H_RESCINDED,               A specified parameter refers to a rescinded shared logical resource.
                                                 (no mappings changed)
                H_R_Parm,                  One or more r-ioba mappings are invalid -- mappings may have changed
                H_Resource,                An attempt was made to make multiple redirected mappings of an RTCE
                                           table entry-- some mappings may have changed.
                H_Hardware                 The function failed due to unrecoverable hardware failure. */

    const uint64 H_PUT_RTCE,        /* Maps RDMA into IOA's TCE table */
    uint64   r-liobn,          /* handle of RDMA RTCE table */
    uint64   r-ioba,           /* IO address per RDMA RTCE table */
    uint64   liobn,            /* Logical I/O Bus Number of server TCE table */
    uint64   ioba,             /* I/O address as seen by server IOA */
    uint64   count);           /* Number of consecutive 4 KB pages to map */
```

Parameters:

- r-liobn: Handle of RDMA RTCE table
- r-ioba: IO address per RDMA RTCE table
- liobn: Logical I/O Bus Number of server TCE table
- ioba: I/O address as seen by server IOA
- count: Number of consecutive 4 KB pages to map

Semantics:

- Validates r-liobn is from the second triple (second window pane) of the server partition's **`"ibm,my-dma-window"`** property, else return H_Parameter.
- Validates r-ioba plus (count * 4 KB) is within range of RTCE table as specified by the window pane as specified by the r-liobn, else return H_Parameter.
- Validates that the TCE table associated with liobn is owned by calling partition, else return H_Parameter.
  - If the Shared Logical Resource option is implemented and the LIOBN, represents a logical resource that has been rescinded by the owner, return H_RESCINDED.
- Validates that ioba plus (count * 4 KB) is within the range of TCE table specified by liobn, else return H_Parameter.

- If the Shared Logical Resource option is implemented and the IOBA represents a logical resource that has been rescinded by the owner, return H_RESCINDED.

- For count entries
  - The following is done in a critical section with respect to updates to the r-ioba entry of the RTCE table TCE
    - Check that the r-ioba entry of the RTCE table contains a valid mapping (this requires having a completed partner connection), else return H_R_Parm with the value of the loop count in R4.
    - Prevent multiple redirected mappings of the same r-ioba: If the r-ioba entry of the RTCE table TCE contains a valid pointer, and if that pointer references a TCE that is a clone of the r-ioba entry of the RTCE table TCE, then return H_Resource with the value of the loop count in R4.
    - Copy the DMA address mapping from the r-ioba entry of the r-liobn RTCE table to the ioba entry of the liobn TCE table and save a pointer to the ioba entry of the liobn TCE table in the r-ioba entry of the r-liobn RTCE table, or in a separate structure associated with the r-liobn RTCE table.
  - End Loop (The critical section lasts for one iteration of the loop)
- Return H_Success

Implementation Note: The PA requires the OS to issue a sync instruction to proceed the signalling of an IOA to start an IO operation involving DMA to guarantee the global visibility of both DMA and TCE data. This hcall() does not include a sync instruction to guarantee global visibility of TCE data and in no way diminishes the requirement for the OS to issue it.

Implementation Note: The execution time for this hcall() is expected to be a linear function of the count parameter. Excessive size of the count parameter may cause an extended delay.

17.2.2.2.2 H_PUT_RTCE_INDIRECT

This hcall() maps “count” number of potentially non-contiguous TCEs in an RTCE to the same number of contiguous IOA TCEs. The H_REMOVE_RTCE hcall() is used to back-out TCEs built with the H_PUT_RTCE_INDIRECT hcall(). See Section 17.2.2.2.3, “H_REMOVE_RTCE,” on page 630 for that hcall().

Syntax:

```
int64 hcall(    /* H_Success,          Expected Return Code
                H_Parameter,           One or more of the parameters are invalid -- no mappings changed.
                H_RESCINDED,           A specified parameter refers to a rescinded shared logical resource/
                                              (No mappings changed)
                H_R_Parm,              One or more r-ioba mappings are invalid -- mappings may have changed
                H_Resource,            An attempt was made to make multiple redirected mappings of an RTCE
                                       table entry-- some mappings may have changed.
                H_Hardware             The function failed due to unrecoverable hardware failure. */
    const uint64 H_PUT_RTCE_INDIRECT, /* Maps RDMA into IOA's TCE table */
    uint64   buff-addr,      /* The Logical Address of a page (4 KB, 4 KB boundary) */
                             /* containing a list of r-ioba to be mapped using the r-liobn RTCE table */
    uint64   r-liobn,        /* handle of RTCE table to be used with r-ioba entries in indirect buffer*/
                             /* (second window pane from server "ibm,my-dma-window")*/
    uint64   liobn,          /* Logical I/O Bus Number of server TCE table */
    uint64   ioba,           /* I/O address as seen by server IOA */
    uint64   count);         /* Number of consecutive IOA bus 4 KB pages to map (number of entries in bufr) */
```

Parameters:

- buff-addr: The Logical Address of a page (4 KB, 4 KB boundary) containing a list of r-ioba to be mapped via using the r-liobn RTCE table
- r-liobn: Handle of RTCE table to be used with r-ioba entries in indirect buffer (second window pane from server **"ibm,my-dma-window"**
- liobn: Logical I/O Bus Number of server TCE table
- ioba: I/O address as seen by server IOA
- count: Number of consecutive IOA bus 4 KB pages to map (number of entries in buffer)

Semantics:

- Validates r-liobn is from the second triple (second window pane) of the server partition's **"ibm,my-dma-window"** property, else return H_Parameter.
- Validates buff-addr points to the beginning of a 4 KB page owned by the calling partition, else return H_Parameter.
  - If the Shared Logical Resource option is implemented and the logical address's page number represents a page that has been rescinded by the owner, return H_RESCINDED.
- Validates that the TCE table associated with liobn is owned by calling partition, else return H_Parameter.
  - If the Shared Logical Resource option is implemented and the LIOBN represents a logical resource that has been rescinded by the owner, return H_RESCINDED.
- Validates that ioba plus (count * 4 KB) is within the range of TCE table specified by liobn, else return H_Parameter.
  - If the Shared Logical Resource option is implemented and the IOBA represents a logical resource that has been rescinded by the owner, return H_RESCINDED.
- If the count field is greater than 512 return H_Parameter.
- Copy (count * 8) bytes from the page specified by buff-addr to a temporary hypervisor page for contents verification and processing (this avoids the problem of the caller changing call by reference values after they are checked).
- For count entries:
  - Validate the r-ioba entry in the temporary page is within range of RTCE table as specified by r-liobn, else place the count number in R4 and return H_R_Parm.
  - End loop
- For count validated entries in the hypervisor's temporary page:
  - The following is done in a critical section with respect to updates to the r-ioba entry of the RTCE
    - Check that the r-ioba entry of the r-liobn RTCE table contains a valid mapping (this requires having a completed partner connection), else return H_R_Parm with the count number in R4.
    - Prevent multiple redirected mappings of the same r-ioba: If the r-ioba entry of the r-liobn RTCE table contains a valid pointer, and if that pointer references a TCE entry that is a clone of the r-ioba entry of the RTCE table, then return H_Resource with the count number in R4.
    - Copy the DMA address mapping from the r-ioba entry of the r-liobn RTCE table to the ioba entry of the liobn TCE table and save a pointer to the ioba entry of the liobn TCE table in the r-ioba entry of the r-liobn RTCE table, or into a separate structure associated with the r-liobn RTCE table.

- End Loop (The critical section lasts for one iteration of the loop)

- Return H_Success

Implementation Note: The PA requires the OS to issue a sync instruction to proceed the signalling of an IOA to start an IO operation involving DMA to guarantee the global visibility of both DMA and TCE data. This hcall() does not include a sync instruction to guarantee global visibility of TCE data and in no way diminishes the requirement for the OS to issue it.

Implementation Note: The execution time for this hcall is expected to be a linear function of the count parameter. Excessive size of the count parameter may cause an extended delay.

17.2.2.2.3 H_REMOVE_RTCE

The H_REMOVE_RTCE hcall() is used to back-out TCEs built with H_PUT_RTCE and H_PUT_RTCE_INDIRECT hcall()s. That is, to remove the TCEs from the IOA TCE table and links put into the RTCE table as a result of the H_PUT_RTCE or H_PUT_RTCE_INDIRECT hcall()s.

Syntax:

```
int64 hcall(    /* H_Success,              Expected Return Code
                H_Parameter,               One or more of the parameters are invalid -- no mappings changed
                H_RESCINDED:               A specified parameter refers to a rescinded shared logical resource.
                                                  (no mappings changed)
                H_Hardware                 The function failed due to unrecoverable hardware failure. */

    const uint64 H_REMOVE_RTCE,     /* Unmaps RDMA from IOA's TCE table */
    uint64   r-liobn,          /* handle of RDMA RTCE table */
    uint64   r-ioba,           /* IO address per RDMA RTCE table */
    uint64   liobn,            /* Logical I/O Bus Number of server TCE table */
    uint64   ioba,             /* I/O address as seen by server IOA */
    uint64   count             /* Number of consecutive 4 KB pages to map */
    uint64   tce-value);       /* TCE value to be put into the IOA TCE (Page Mapping and Control bits will be */
                               /* set to "Page fault (no access) by the hcall before replacing the IOA TCE) */
```

Parameters:

- r-liobn: Handle of RDMA RTCE table
- r-ioba: IO address per RDMA RTCE table
- liobn: Logical I/O Bus Number of server TCE table
- ioba: I/O address as seen by server IOA
- count: Number of consecutive 4 KB pages to unmap
- tce-value: TCE value to be put into the IOA TCE(s) after setting the "Page Mapping and Control" bits to "Page fault (no access)".

Semantics:

- Validates r-liobn is from the second triple (second window pane) of the server partition's **"ibm,my-dma-window"** property, else return H_Parameter.
- Validates r-ioba plus (count * 4 KB) is within range of RTCE table as specified by the window pane as specified by the r-liobn, else return H_Parameter.
- Validates that the TCE table associated with liobn is owned by calling partition, else return H_Parameter.

- If the Shared Logical Resource option is implemented and the LIOBN, represents a logical resource that has been rescinded by the owner, return H_RESCINDED.

- Validates that ioba plus (count * 4 KB) is within the range of TCE table specified by liobn, else return H_Parameter.
  - If the Shared Logical Resource option is implemented and the IOBA represents a logical resource that has been rescinded by the owner, return H_RESCINDED.
- For count entries
  - The following is done in a critical section with respect to updates the r-ioba entry of the RTCE table TCE
    - If it exists, invalidate the pointer in the r-ioba entry of the r-liobn RTCE table (or in a separate structure associated with the r-liobn RTCE table).
    - Replace the ioba entry of the liobn TCE table with tce-value after setting the “Page Mapping and Control” bits to “Page fault (no access)”.
  - End Loop (The critical section lasts for one iteration of the loop)
- Return H_Success

Implementation Note: The execution time for this hcall() is expected to be a linear function of the count parameter. Excessive size of the count parameter may cause an extended delay.

17.2.2.2.4 Redirected RDMA TCE Recovery and In-Flight DMA

There are certain error or error recovery scenarios that may attempt to unmap a TCE in an IOA’s TCE table prior to the completion of the operation which setup the TCE. For example:

- A client attempts to H_PUT_TCE to its DMA window pane, which is mapped to the second window pane of the server’s DMA window, and the TCE in the RTCE table which is the target of the H_PUT_TCE already points to a valid TCE in an IOA’s TCE table.
- A client attempts to H_FREE_CRQ and the server’s second window pane for that virtual IOA contains a TCE which points to a valid TCE in an IOA’s TCE table.
- A client partition attempts to reboot (which essentially is an H_FREE_CRQ).
- A server attempts to H_FREE_CRQ and the server’s second window pane for that virtual IOA contains a TCE which points to a valid TCE in an IOA’s TCE table.

In such error and error recovery situations, the hypervisor attempts to prevent the changing of an IOAs TCE to a value that would cause a non-recoverable IOA error. One method that the hypervisor may use to accomplish this is that on a TCE invalidation operation, set the value of the read and write enable bits in the TCE to allow DMA writes but not reads, and to change the real page number in the TCE to target a dummy page. In this case the IOA receives an error (Target Abort) on attempts to read, while DMA writes (which were for a defunct operation) are silently dropped. This works well when all the following are true:

- The platform supports separate TCE read and write enable bits in the TCE
- EEH is enabled and the DD can recover from the MMIO Stopped and DMA Stopped states
- The IOA and the IOA’s DD can recover gracefully from Target Aborts (which are received on a read to a page where the read enable bit is off)

If these conditions are not true then the hypervisor will need to try to prevent or delay invalidation of the TCEs. The H_Resource return from the H_FREE_CRQ, H_PUT_TCE, H_PUT_TCE_INDIRECT, and H_STUFF_TCE can be used to hold-off the invalidation until which time the IOA can complete the operation and the server can invalidate the

IOA's TCE. In addition, the Bit Bucket Allowed LIOBN attribute and the H_LIOBN_ATTRIBUTES hcall can be used to help enhance the recoverability in these error scenarios (see Section 17.2.2.2.5, "LIOBN Attributes," on page 632 and Section 17.2.2.2.6, "H_LIOBN_ATTRIBUTES," on page 632 for more information).

17.2.2.2.5 LIOBN Attributes

There are certain LIOBN attributes that are made visible to and can be manipulated by partition software. The H_LIOBN_ATTRIBUTES hcall is used to read and modify the attributes (see Section 17.2.2.2.6, "H_LIOBN_ATTRIBUTES," on page 632). Table 233, "LIOBN Attributes," on page 632 defines the attributes that are visible and manipulatable.

Table 233. LIOBN Attributes

Bit(s)	Field Name	Definition
0-62	Reserved	
63	Bit Bucket Allowed	1: For an indirect IOA TCE invalidation operation (that is, via an operation other than an H_PUT_TCE directly to the TCE by the partition owning the IOA), the platform may set the value of the read and write enable bits in the TCE to allow DMA writes but not reads and change the real page number in the TCE to target a dummy page (the IOA receives an error (Target Abort) on attempts to read, while DMA writes (which were for a defunct operation) are silently dropped). 0: The platform must reasonably attempt to prevent an indirect (that is, via an operation other than an H_PUT_TCE directly to the TCE by the partition owning the IOA) modification an IOA's valid TCE so that a possible in-flight DMA does not cause a non-recoverable error. **Software Implementation Notes:** 1. The results of changing this field when there are valid TCEs for the LIOBN may produce unexpected results. The hypervisor is not required to prevent such an operation. Therefore, the H_LIOBN_ATTRIBUTES call to change the value of this field should be made when there are no valid TCEs in the table for the IOA. 2. This field may be implemented but not changeable (the actual value will be returned in R4 as a result of the H_LIOBN_ATTRIBUTES hcall() regardless, with a status of H_Constrained if not changeable).

17.2.2.2.6 H_LIOBN_ATTRIBUTES

R1–17.2.2.2.6–1. If the H_LIOBN_ATTRIBUTES hcall is implemented, then it must implement the attributes as they are defined in Table 233, "LIOBN Attributes," on page 632 and the syntax and semantics as defined in Section 17.2.2.2.6, "H_LIOBN_ATTRIBUTES," on page 632.

R1–17.2.2.2.6–2. The H_LIOBN_ATTRIBUTES hcall must ignore bits in the set-mask and reset-mask which are not implemented and must process as an exception those which cannot be changed (H_Constrained returned), and must return the following for the LIOBN Attributes in R4:

a. A value of 0 for unimplemented bit positions.

b. The resultant field values for implemented fields.

Syntax:

```
uint64      /*  H_Success       Expected return code */
            /*  H_Parameter     One or more parameters were in error */
            /*  H_Constrained   One or more parameters were not changeable to the value requested*/
            /*                  The result was constrained to a legitimate value for the implementation*/
            /*  H_Hardware      Operation failed because of hardware error*/
  hcall (   const uint64 H_LIOBN_ATTRIBUTES,/* Returns in R4 the resulting LIOBN Attributes*/
            uint64    liobn,    /* The LIOBN on which to perform this operation*/
```

```
uint64  reset-mask,     /* Mask of Attribute bits to be reset*/
uint64  set-mask);      /* Mask of Attribute bits to be set*/
```

Parameters:

liobn: The LIOBN on which this Attribute modification is to be performed.

reset-mask: The bit-significant mask of bits to be reset in the LIOBN's Attributes (the reset-mask bit definition aligns with the bit definition of the LIOBN's Attributes, as defined in Table 233, "LIOBN Attributes," on page 632). The complement of the reset-mask is ANDed with the LIOBN's Attributes, prior to applying the set-mask. See semantics for more details on any field-specific actions needed during the reset operations. If a particular field position in the LIOBN Attributes is not implemented, then the corresponding bit(s) in the reset-mask are ignored.

set-mask: The bit-significant mask of bits to be set in the LIOBN's Attributes (the set-mask bit definition aligns with the bit definition of the LIOBN's Attributes, as defined in Table 233, "LIOBN Attributes," on page 632). The set-mask is ORed with the LIOBN's Attributes, after to applying the reset-mask. See semantics for more details on any field-specific actions needed during the set operations. If a particular field position in the LIOBN Attributes is not implemented, then the corresponding bit(s) in the set-mask are ignored.

Semantics:

- Validate that liobn belongs to the partition, else H_Parameter.
- If the Bit Bucket Allowed field of the specified LIOBN's Attributes is implemented and changeable, then set it to the result of:
  Bit Bucket Allowed field contents ANDed with the complement of the corresponding bits of the reset-mask and then ORed with the corresponding bits of the set-mask.
- Load R4 with the value of the LIOBN's Attributes, with any unimplemented bits set to 0, and if all requested changes were made, then return H_Success, otherwise return H_Constrained.

17.2.2.2.7 Extensions to Other hcall()s for Redirected RDMA

17.2.2.2.7.1 H_PUT_TCE, H_PUT_TCE_INDIRECT, and H_STUFF_TCE

These hcall()s are only valid for the first window pane of the **"ibm,my-dma-window"** property. See Section 17.2.1.2, "RTCE Table and Properties of the Children of the /vdevice Node," on page 601 for information about window pane types.

The following are extensions that apply to the H_PUT_TCE, H_PUT_TCE_INDIRECT, and H_STUFF_TCE hcall()s in their use against an RTCE table.

Recognize the validated (owned by the calling partition, else H-Parameter) LIOBN as referring to a RTCE table (first window pane) and access accordingly:

- If the TCE is not from the first triple (first window pane) of the calling partition's **"ibm,my-dma-window"** property, return H_Parameter.
- If the TCE is not currently in use: Clear/invalidate the TCE copy pointer and enter the TCE mapping per the input parameters to the hcall().
- If the TCE contains a valid mapping and the TCE copy pointer is invalid: Enter the TCE mapping per the input parameters to the hcall().
- If the TCE contains a valid mapping and the TCE copy pointer references a TCE that does not contain a valid copy of the previous mapping in the TCE: Clear/invalidate the TCE copy pointer and enter the TCE mapping per the input parameters to the hcall().

- If the TCE contains a valid mapping and the TCE copy pointer references a TCE that does contain a valid copy of the previous mapping in the TCE, then:
  - If the Bit Bucket Allowed Attribute of the LIOBN containing the TCE is a 1, invalidate the copied TCE and enter the TCE mapping per the input parameters to the hcall().
  - If the Bit Bucket Allowed Attribute of the LIOBN containing the TCE is a 0, then return H_Resource or perform some other platform-specific error recovery.

17.2.2.2.7.2 H_MIGRATE_DMA

Check that the pages referenced by the TCEs specified in the mappings to be migrated belong to the calling partition, else H_Parameter.

If the mapping being migrated is via an RTCE table (that is, LIOBN points to an RTCE table), then follow the valid re-directed TCE pointer and migrate the redirected page (if the redirected TCE mapping is still a clone of the original RTCE table entry).

If the mapping being migrated is via an RTCE table and if the RTCE table TCEs were built with the H_MASS_MAP_TCE hcall(), then expand each mass mapped area into smaller 4 KB granularities, as necessary to avoid performance and locking issues, during the migration process.

Insert checking, and potentially delays, to allow IOAs to make forward progress between successive DMA disables caused by multiple partner partitions making simultaneous uncoordinated calls to H_MIGRATE_DMA targeting the same IOA.

17.2.2.3 Subordinate Command/Response Queue (Sub-CRQ)

The Sub-CRQ facility is used in conjunction with the CRQ facility, for some virtual IOA types, when more than one queue is needed for the virtual IOA. For information on the CRQ facility, see Section 17.2.2.1, “Command/Response Queue (CRQ),” on page 621. For information on which virtual IOAs may use the Sub-CRQ facilities, see the applicable sections for the virtual IOAs. See Table 234, “CRQ and Sub-CRQ Comparison,” on page 634 for a comparison of the differences in the queue structures between CRQs and Sub-CRQs. In addition to the hcall()s specified in Table 234, all of the following hcall()s and RTAS calls are applicable to both CRQs and Sub-CRQs:

- H_XIRR
- H_EOI
- ibm,int-on
- ibm,int-off
- ibm,set-xive
- ibm,get-xive

Table 234. CRQ and Sub-CRQ Comparison

Characteristic	CRQ	Sub-CRQ
Queue entry size	16	32
Transport and initialization events	Applicable	Not applicable (coordinated through the CRQ that is associated with the Sub-CRQ)
Registration	H_REG_CRQ	H_REG_SUB_CRQ

Table 234. CRQ and Sub-CRQ Comparison *(Continued)*

Characteristic	CRQ	Sub-CRQ
Deregistration	H_FREE_CRQ	H_FREE_SUB_CRQ Note: H_FREE_CRQ for the associated CRQ implicitly deregisters the associated Sub-CRQs
Enable	H_ENABLE_CRQ	Not applicable
Interrupt number	Obtained from **"interrupts"** property	Obtained from H_REG_SUB_CRQ
Interrupt enable/disable	H_VIO_SIGNAL	H_VIOCTL subfunction[a]
hcall() used to place entry on queue	H_SEND_CRQ	H_SEND_SUB_CRQ H_SEND_SUB_CRQ_INDIRECT
Number of queues per virtual IOA	One	Zero or more, depending on virtual IOA architecture, implementation, and client/server negotiation

a. For virtual IOAs that define the use of Sub-CRQs, the interrupt associated with the CRQ, as defined by the **"interrupts"** property in the OF device tree, may be enabled or disabled with either the H_VIOCTL or the H_VIO_SIGNAL hcall(). The CRQ interrupt associated with a CRQ of a virtual IOA that does not define the use of Sub-CRQs should be enabled and disabled by use of the H_VIO_SIGNAL hcall().

17.2.2.3.1 Sub-CRQ Format and Registration

Each Sub-CRQ is built of one or more 4 KB pages aligned on a 4 KB boundary within partition memory, and is organized as a circular buffer of 32 byte long elements. Each queue is mapped into contiguous I/O addresses via the TCE mechanism and RTCE table (first window pane). The I/O address and length of each queue is registered by the process defined in Section 17.2.3.3.1, "Sub-CRQ Format and Registration," on page 646. This registration process tells the hypervisor where to find the virtual IOA's Sub-CRQ(s).

17.2.2.3.2 Sub-CRQ Entry Format

Each Sub-CRQ entry consists of a 32 byte element. The first byte of a Sub-CRQ entry is the Header byte and is defined in Table 235, "Sub-CRQ Entry Header Byte Values," on page 635.

Table 235. Sub-CRQ Entry Header Byte Values

Header Value	Description
0	Element is unused -- all other bytes in the element are undefined.
0x01 - 0x7F	Reserved.
0x80	Valid Command/Response entry.
0x81 - 0xFF	Reserved.

The platform (transport mechanism) ignores the contents of all non-header bytes in all Sub-CRQ entries.

The operational state of any Sub-CRQs follows the operational state of the CRQ to which the Sub-CRQ is associated. That is, the CRQ transport is required to be operational in order for any associated Sub-CRQs to be operational (for example, if an H_SEND_CRQ hcall() would not succeed due to any reason other than lack of space is available in the CRQ, then an H_SEND_SUB_CRQ or H_SEND_SUB_CRQ_INDIRECT hcall() to the associated Sub-CRQ would also fail). Hence, the Sub-CRQ transport does not implement the transport and initialization events that are implemented by the CRQ facility.

17.2.2.3.3 Sub-CRQ Entry Processing

During the Sub-CRQ registration (H_REG_SUB_CRQ), the platform firmware sets all the header bytes of the Sub-CRQ being registered to zero (entry invalid). After registration, the first valid entry is placed in the first element and the process proceeds to the end of the queue and then wraps around to the first entry again (given that the entry has been subsequently marked as invalid). This allows both the partition software and transport firmware to maintain independent pointers to the next element they will be respectively using.

A sender uses an H_SEND_SUB_CRQ hcall() to enter one 32 byte message on its partner's Sub-CRQ. Prior to enqueueing an entry on the Sub-CRQ, the platform first checks if the session to the partner's associate CRQ is open, and there is a enough free space on the Sub-CRQ, if not, it returns an error. If the checks succeed, the contents of the message is copied into the next free queue element, potentially notifying the receiver, and returns a successful status to the caller. The caller may also insert more than one entry on the queue with one hcall() using H_SEND_SUB_CRQ_INDIRECT. Use of this hcall() requires that there be enough space on the queue for all the entries, otherwise none of the entries are placed onto the Sub-CRQ.

At the receiver's option, it may be notified via an interrupt when an element is enqueued to its Sub-CRQ. See "Sub-CRQ Facility Interrupt Notification" on page 636.

When the receiver has finished processing a Sub-CRQ entry, it writes the header to the value 0x00 to invalidate the entry and free it for future entries.

Should the receiver wish to terminate or reset the communication channel it deregisters the Sub-CRQ (H_FREE_SUB_CRQ), and if it needs to re-establish communications, proceeds to register (H_REG_SUB_CRQ) either the same or different section of memory as the new queue, with the queue pointers reset to the first entry. Deregistering a CRQ (H_FREE_CRQ) is an implicit deregistration of any Sub-CRQs associated with the CRQ.

17.2.2.3.4 Sub-CRQ Facility Interrupt Notification

The receiver can set the virtual interrupt associated with its Sub-CRQ to one of two modes. These are:

1. Disabled (an enqueued interrupt is not signaled).
2. Enabled (an enqueued interrupt is signaled on every enqueue).

Note: An enqueue is considered a pulse not a level. The pulse then sets the memory element within the emulated interrupt source controller. This allows the resetting of the interrupt condition by simply issuing the H_EOI hcall() as is done with the PCI MSI architecture rather than having to do an explicit interrupt reset as in the case with PCI Level Sensitive Interrupt (LSI) architecture.

The interrupt mechanism is capable of presenting only one interrupt signal at a time from any given interrupt source. Therefore, no additional interrupts from a given source are ever signaled until the previous interrupt has been processed through to the issuance of an H_EOI hcall(). Specifically, even if the interrupt mode is enabled, the effect is to interrupt on an empty to non-empty transition of the queue. However, as with any asynchronous posting operation race conditions are to be expected. That is, an enqueue can happen in a window around the H_EOI hcall(). Therefore, the receiver should poll the Sub-CRQ (that is, look at the header byte of the next queue entry to see if the entry is valid) after an H_EOI to prevent losing initiative.

The hcall() used to enable and disable this Sub-CRQ interrupt notification is H_VIO_SIGNAL (see Section 17.2.1.3, "VIO Interrupt Control," on page 602).

17.2.2.3.5 Extensions to Other hcall()s for Sub-CRQ

17.2.2.3.5.1 H_MIGRATE_DMA

Since Sub-CRQs are RTCE table mapped, the H_MIGRATE_DMA hcall() may be requested to move a page that is part of a Sub-CRQ. The OS owner of the queue is responsible for preventing its processors from modifying the page

during the migrate operation (as is standard practice with this hcall()), however, the H_MIGRATE_DMA hcall() serializes with the Sub-CRQ hcall()s to direct new elements to the migrated target page.

17.2.2.3.5.2 H_XIRR, H_EOI

The Sub-CRQ facility utilizes a virtual interrupt source number to notify the queue owner of new element enqueues. The standard H_XIRR and H_EOI hcall()s are extended to support this virtual interrupt mechanism, emulating the standard PowerPC Interrupt hardware with respect to the virtual interrupt source number.

17.2.2.3.6 Sub-CRQ Facility Requirements

R1–17.2.2.3.6–1. For the Sub-CRQ facility: The platform must implement the Sub-CRQ as specified in Section 17.2.2.1, “Command/Response Queue (CRQ),” on page 621.

R1–17.2.2.3.6–2. For the Sub-CRQ facility: The platform must start enqueueing Commands/Responses to the newly registered Sub-CRQ starting at offset zero and proceeding as in a circular buffer, each entry being 32 byte aligned.

R1–17.2.2.3.6–3. For the Sub-CRQ facility: The platform must enqueue Commands/Responses only if the 32 byte entry is free (header byte contains 0x00), else the enqueue operation fails.

R1–17.2.2.3.6–4. For the Sub-CRQ facility: The first byte of a Sub-CRQ entry must be the Header byte and must be as defined in Table 235, “Sub-CRQ Entry Header Byte Values,” on page 635.

R1–17.2.2.3.6–5. For the Sub-CRQ facility option: Platforms that implement the H_MIGRATE_DMA hcall() must implement that function for pages mapped for use by the Sub-CRQ.

R1–17.2.2.3.6–6. For the Sub-CRQ facility: The platforms must emulate the standard PowerPC External Interrupt Architecture for the interrupt source numbers associated with the virtual devices via the standard RTAS and hypervisor interrupt calls and must extend H_XIRR and H_EOI hcall()s as appropriate for Sub-CRQ interrupts.

17.2.3 Partition Managed Class - Synchronous Infrastructure

The architectural intent of the Synchronous VIO infrastructure is for platforms where the communicating partitions are under the control of the same hypervisor. Operations between the partitions are via synchronous hcall() operations. The Synchronous VIO infrastructure defines three options:

- Reliable Command/Response Transport option (see Section 17.2.3.1, “Reliable Command/Response Transport Option,” on page 637
- Subordinate CRQ Transport option (see Section 17.2.3.3, “Subordinate CRQ Transport Option,” on page 645
- Logical Remote DMA (LRDMA) option (see Section 17.2.3.2, “Logical Remote DMA (LRDMA) Option,” on page 642)

17.2.3.1 Reliable Command/Response Transport Option

For the synchronous infrastructure, the CRQ facility defined in Section 17.2.2.1, “Command/Response Queue (CRQ),” on page 621 is implemented via the Reliable Command/Response Transport option. The synchronous nature of this infrastructure allows for the capability to immediately (synchronously) notify the sender of the message whether the message was delivered successfully or not.

17.2.3.1.1 Reliable CRQ Format and Registration

The format of the CRQ is as defined in Section 17.2.2.1, “Command/Response Queue (CRQ),” on page 621.

The I/O address and length of the queue are registered using the H_REG_CRQ hcall(). See Section 17.2.3.1.5.1, “H_REG_CRQ,” on page 638.

17.2.3.1.2 Reliable CRQ Entry Format

See Section 17.2.2.1.2, “CRQ Entry Format,” on page 621.

17.2.3.1.3 Reliable CRQ Entry Processing

A sender uses the H_SEND_CRQ hcall() to enter a 16 byte message on its partner’s queue. The hcall() takes the entire message as input parameters in two registers. See Section 17.2.3.1.5.3, “H_SEND_CRQ,” on page 640.

17.2.3.1.4 Reliable Command/Response Transport Interrupt Notification

The receiver can enable and disable the virtual interrupt associated with its CRQ. See Section 17.2.1.3, “VIO Interrupt Control,” on page 602.

17.2.3.1.5 Reliable Command/Response Transport hcall()s

The H_REG_CRQ and H_FREE_CRQ hcall()s are used by both client and server virtual IOA device drivers. It is the architectural intent that the hypervisor maintains a connection control structure for each defined partner/server connection. The H_REG_CRQ and its corresponding H_FREE_CRQ register and deregister partition resources with that connection control structure. However, there are several conditions that can arise architecturally with this connection process (the design of an implementation may preclude some of these conditions).

- The association connection to the partner virtual IOA not being defined (H_Not_Found). The CRQ registration function fails if the CRQ is not registered with the hypervisor.
- The partner virtual IOA may not have registered its CRQ (H_Closed). The CRQ is registered with the hypervisor and the connection. However, the connection is incomplete because their partner has not registered.
- The partner virtual IOA may be already connected to another partner virtual IOA (H_Resource). The CRQ registration function fails if the CRQ is not registered with the hypervisor or the connection.

The reaction of the virtual IOA device driver to these conditions is somewhat different depending upon the calling device driver being for a client or server IOA. Server IOAs in many cases register prior to their partner IOAs since they are servers and subsequently wait for service requests from their clients. Therefore, the H_Closed return code is to be expected when the DD’s CRQ has been registered with the connection and is just waiting for the partner to register. Should a partner DD register its CRQ in the future, higher level protocol messages (via the Initialization Command/Response CRQ entry) can notify the server DD when the connection is established. If a client IOA registers and receives a return code of H_Closed, it may choose to deregister the CRQ and fail since the client IOA would not be in a position to successfully send service requests using the CRQ facility, or it may wait and rely upon higher level CRQ messages (via the Initialization Command/Response CRQ entry) to tell it when its partner has registered. The reaction of a virtual IOA DDs to H_Not_Found and H_Resource are dependent upon the functionality of higher level platform and system management policies. While the current registration has failed, higher level system and or platform management actions may allow a future registration request to succeed.

When registration succeeds, an association is made between the partner partition’s LIOBN (RTCE table) and the second window pane of the server partition. This association is dropped when either partner deregisters or terminates. However, on deregistration or termination, the RTCE tables associated with the local partition (first window pane) remain intact for that partition (see Requirement R1–17.2.1.4–15).

17.2.3.1.5.1 H_REG_CRQ

This hcall() registers the RTCE table mapped memory that contains the CRQ.

Syntax:

```
int64                                   /*H_Success: Registration and Connection completed*/
                                        /*H_Parameter: Failed due to Invalid Parameter */
                                        /*H_Not_Found: Failed due to undefined partner connection.*/
                                        /*H_Closed: Registration completed partner not connected.
                                        /*H_Resource: Failed due to busy connection to partner.
                                        /*H_Hardware: Function failed due to hardware error */
    hcall (const int64 H_REG_CRQ,       /* Function Code */
            uint64 unit-address,        /* As specified in the Virtual IOA's device tree node */
            uint64 queue,               /* I/O address of a receive queue (4 KB aligned)*/
            uint64 len)                 /* Length of the receive queue (multiple of 4 KB) */
```

Parameters:

- unit-address: Unit Address per device tree node **"reg"** property
- queue: I/O address (offset into the RTCE table) of the CRQ buffer (starting on a 4 KB boundary).
- len: Length of the CRQ in bytes (a multiple of 4 KB)

Semantics:

- Validate unit-address, else H_Parameter
- Validate queue, which is the I/O address of the CRQ (I/O addresses for entire buffer length starting at the specified I/O address are translated by the RTCE table, is 4 KB aligned, and length, len, is a multiple of 4 KB), else H_Parameter
- Validate that there is an authorized connection to another partition associated with the Unit Address, else H_Not_Found.
- Validate that the authorized connection to another partition associated with the Unit Address is free, else H_Resource.
- Initialize the CRQ enqueue pointer and length variables. These variables are kept in terms of I/O addresses so that page migration works and any remapping of TCEs is effective.
- Disable CRQ interrupts.
- Allow for Logical Remote DMA, when applicable, with associated partner partition when partner registers.
- If partner is already registered, then return H_Success, else return H_Closed.

17.2.3.1.5.2 H_FREE_CRQ

This hcall() deregisters the RTCE table mapped memory that contains the CRQ. In addition, if there are any Sub-CRQs associated with the CRQ, the H_FREE_CRQ has the effect of releasing the Sub-CRQs.

Syntax:

```
int64                                   /* H_Success, H_Parameter, H_Resource, H_Hardware, H_Busy, */
                                        /* H_LongBusyOrder1mSec, H_LongBusyOrder10mSec,*/
    hcall (const int64 H_FREE_CRQ,      /* Function Code */
            uint64 unit-address)        /* As specified in the Virtual IOA's device tree node */
```

Parameters:

- unit-address: Unit Address per device tree node **"reg"** property

Semantics:

- Validate unit-address, else H_Parameter
- Mark the connection to the associated partner partition as closed (so that send hcall()s from the partner partition fail).
- Mark the CRQ enqueue pointer and length variables as invalid.
- For any and all Sub-CRQs associated with the CRQ, do the following:
  - Mark the connection to the associated partner partition as closed for the Sub-CRQ (so that send hcall()s from the partner partition fail).
  - Mark the Sub-CRQ enqueue pointer and length variables for the Sub-CRQ as invalid.
  - Disable Sub-CRQ interrupts for the Sub-CRQ.
- Disable CRQ interrupts.
- If there exists any Redirected TCEs in the local TCE tables associated with this Virtual IOA, and all of those tables have a Bit Bucket Allowed attribute of 1, then Disable Logical Remote DMA with associated partner partition, if enabled, invalidating any Redirected TCEs in the local TCE tables (for information on invalidation of TCEs, see Section 17.2.2.2.4, “Redirected RDMA TCE Recovery and In-Flight DMA,” on page 631).
- If there exists any Redirected TCEs in the local TCE tables associated with this Virtual IOA, and any of those tables have a Bit Bucket Allowed attribute of 0, then return H_Resource or perform some other platform-specific error recovery.
- Send partner terminated message to partner queue (if it is still registered), overlaying the last valid entry in the queue if the CRQ is full.
- Return H_Success.

Implementation Note: If the hypervisor returns an H_Busy, H_LongBusyOrder1mSec, or H_LongBusyOrder10mSec, software must call H_FREE_CRQ again with the same parameters. Software may choose to treat H_LongBusyOrder1mSec and H_LongBusyOrder10mSec the same as H_Busy. The hypervisor, prior to returning H_Busy, H_LongBusyOrder1mSec, or H_LongBusyOrder10mSec, will have placed the virtual adapter in a state that will cause it to not accept any new work nor surface any new virtual interrupts (no new entries will be placed on the CRQ).

17.2.3.1.5.3 H_SEND_CRQ

This hcall() sends one 16 byte entry to the partner partition’s registered CRQ.

Syntax:

```
int64                                   /* H_Success, H_Parameter, H_Dropped, H_Hardware, H_Closed*/
    hcall (const int64 H_SEND_CRQ,      /* Function Code */
            uint64 unit-address,        /* As specified in the Virtual IOA’s device tree node */
            uint64 msg-high,            /* High order 8 bytes of message starting with the header and format bytes*/
            uint64 msg-low)             /* Low order 8 bytes of message */
```

Parameters:

- unit-addr: Unit Address per device tree node **`“reg”`** property

- msg-high:
  - header: high order bit is on -- header of value 0xFF is reserved for transport error and is invalid for input.
  - format: not checked by the firmware.
- msg-low: not checked by the firmware -- should be consistent with the definition of the format byte.

Semantics:

- Validate the Unit Address, else return H_Parameter
- Validate that the msg header byte has its high order bit on and that it is not = 0xFF, else return H_Parameter.
- Validate that there is an authorized connection to another partition associated with the Unit Address and that the associated CRQ is enabled, else return H_Closed.
- Enter Critical Section on target CRQ
  - Validate that there is room on the receive queue for the message and allocate that message, else exit critical Section and return H_Dropped.
  - Store msg-low into the second 8 bytes of the allocated queue element.
  - Store order barrier
  - Store msg-high into the first 8 bytes of the allocated queue element (setting the header valid bit.)
- Exit Critical Section
- If receiver queue interrupt mode == enabled, then signal interrupt
- Return H_Success.

17.2.3.1.5.4 H_ENABLE_CRQ

This hcall() explicitly enables a CRQ that has been disabled due to a Partner partition suspended transport event. As a side effect of this hcall(), all pages that are mapped via the logical TCE table associated with the first pane of **`"ibm,my-dma-window"`** property of the associated virtual IOA are restored prior to successful completion of the hcall(). It is the architectural intent that this hcall() is made while the logical TCE contains mappings for all the pages that will be involved in the recovery of the outstanding I/O operations at the time of the partition migration. Further, it is the architectural intent that this hcall() is made from a processing context that can handle the expected busy wait return code without blocking the processor.

Syntax:

```
int64                                   /* H_Success: CRQ enabled and TCE mapped pages restored */
                                        /* H_Parameter: Invalid unit-address */
                                        /* H_Hardware: A hardware error prevented the function */
                                        /* H_LongBusyOrder10mSec: wait for asynchronous processing */
                                        /* H_IN_PROGRESS: more processing required to complete -- call again */
    hcall (const int64 H_ENABLE_CRQ, /* Function Code */
            uint64 unit-address)        /* As specified in the Virtual IOA's device tree node */
```

Parameters:

- unit-addr: Unit Address per device tree node **`"reg"`** property

Semantics:

- Validate the Unit Address, else return H_Parameter
- Test that all pages mapped through the logical TCE table associated with the first pane of the **"ibm,my-dma-window"** property associated with the unit-address parameter are present; else return H_LongBusyOrder10mSec.
- Set the status of the CRQ associated with the unit-address parameter to enabled.
- Return H_Success.

17.2.3.1.6 Reliable Command/Response Transport Option Requirements

R1–17.2.3.1.6–1. For the Reliable Command/Response Transport option: The platform must implement the CRQ facility, as defined in Section 17.2.2.1, "Command/Response Queue (CRQ)," on page 621.

R1–17.2.3.1.6–2. For the Reliable Command/Response Transport option: The platform must implement the H_REG_CRQ hcall(). See Section 17.2.3.1.5.1, "H_REG_CRQ," on page 638.

R1–17.2.3.1.6–3. For the Reliable Command/Response Transport option: The platform must implement the H_FREE_CRQ hcall(). See Section 17.2.3.1.5.2, "H_FREE_CRQ," on page 639.

R1–17.2.3.1.6–4. For the Reliable Command/Response Transport option: The platform must implement the H_SEND_CRQ hcall(). See Section 17.2.3.1.5.3, "H_SEND_CRQ," on page 640.

R1–17.2.3.1.6–5. For the Reliable Command/Response Transport option: The platform must implement the H_ENABLE_CRQ hcall(). See Section 17.2.3.1.5.4, "H_ENABLE_CRQ," on page 641.

17.2.3.2 Logical Remote DMA (LRDMA) Option

The Logical Remote Direct Memory Access (LRDMA) option allows a server partition to securely target memory pages within a partner partition for VIO operations.

This architecture defines two modes of RDMA

- *Copy RDMA* is used to have the hypervisor copy data between a buffer in the server partition's memory and a buffer in the partner partition's memory. See Section 17.2.3.2.1, "Copy RDMA," on page 642 for more information on Copy RDMA with respect to LRDMA.
- *Redirected RDMA* allows for a server partition to securely target its I/O adapter's DMA operations directly at the memory pages of the partner partition. The platform overhead of Copy RDMA is generally greater than Redirected RDMA, but this overhead may be offset if the server partition's DMA buffer is being used as a data cache for multiple VIO operations. See Section 17.2.2.2, "Redirected RDMA (Using H_PUT_RTCE, and H_PUT_RTCE_INDIRECT)," on page 625 for more information on Redirected RDMA with respect to LRDMA.

The mapping between the LIOBN in the second pane of a server virtual IOA's **"ibm,my-dma-window"** property and the corresponding partner IOA's RTCE table is made when the CRQ successfully completes registration. The partner partition is not aware if the server partition is using Copy RDMA or Redirected RDMA. The server partition uses the Logical RDMA mode that best suits its needs for a given VIO operation. See Section 17.2.1.2, "RTCE Table and Properties of the Children of the /vdevice Node," on page 601 for more information on RTCE tables.

17.2.3.2.1 Copy RDMA

The Copy RDMA hcall()s are used to request that the hypervisor move data between partitions. The specific implementation is optimized to the platform's hardware features. There are calls for when both source and destination buf-

fers are RTCE table mapped (H_COPY_DMA) and when only the remote buffers are mapped (H_WRITE_RDMA and H_READ_RDMA).

17.2.3.2.1.1 H_COPY_RDMA

This hcall() copies data from an RTCE table mapped buffer in one partition to an RTCE table mapped buffer in another partition, with the length of the transfer being specified by the transfer length parameter in the hcall(). The **"ibm,max-virtual-dma-size"** property, if it exists (in the **/vdevice** node), specifies the maximum length of the transfer (minimum value of this property is 128 KB).

Syntax:

```
int64 hcall(    /* H_Success, H_Parameter, H_S_Parm, H_D_Parm, H_Permission, H_Hardware */
    const uint64 H_COPY_RDMA,       /* Performs Copy RDMA */
    int64    len,               /* Length of transfer */
    uint64   s-liobn,           /* LIOBN (RTCE table handle) of V-DMA source buffer */
    uint64   s-ioba,            /* IO address of V-DMA source buffer */
    uint64   d-liobn,           /* LIOBN (RTCE table handle) of V-DMA destination buffer */
    uint64   d-ioba);           /* I/O address of V-DMA destination buffer */
```

Parameters:

- len: Length of transfer (length not to exceed the value in the **"ibm,max-virtual-dma-size"** property, if it exists)
- s-liobn: LIOBN (RTCE table handle) of V-DMA source buffer
- s-ioba: IO address of V-DMA source buffer
- d-liobn: LIOBN (RTCE table handle) of V-DMA destination buffer
- d-ioba: I/O address of V-DMA destination buffer

Semantics:

- Serialize access to RTCE tables with H_MIGRATE_DMA.
- If the **"ibm,max-virtual-dma-size"** property exist in the **/vdevice** node of the device tree, then if the value of len is greater than the value of this property, return H_Parameter.
- Source and destination LIOBNs are checked for authorization per the **"ibm,my-dma-window"** property, else return H_S_Parm or H_D_Parm, respectively.
- Source and destination ioba's and length are checked for valid ranges per the **"ibm,my-dma-window"** property, else return H_S_Parm or H_D_Parm, respectively.
- The access bits of the associated TCEs are checked for authorization, else return H_Permission.
- Copy len number of bytes from the buffer starting at the specified source address to the buffer starting at the specified destination address, then return H_Success.

17.2.3.2.1.2 H_WRITE_RDMA

This hcall() copies up to 48 bytes of data from a set of input parameters to an RTCE table mapped buffer in another partition.

Syntax:

```
int64 hcall(    /* H_Success, H_Parameter, H_D_Parm, H_Permission, H_Hardware */
      const uint64 H_WRITE_RDMA,      /* Performs Copy RDMA */
      int64     len,                  /* Length of transfer */
      uint64    d-liobn,              /* LIOBN (RTCE table handle) of V-DMA destination buffer */
      uint64    d-ioba,               /* I/O address of V-DMA destination buffer */
      uint64    data1,                /* The source data is provided via input parameters */
      uint64    data2,
      uint64    data3,
      uint64    data4,
      uint64    data5,
      uint64    data6);
```

Parameters:

- len: Length of transfer
- d-liobn: LIOBN (RTCE table handle) of V-DMA destination buffer
- d-ioba: I/O address of V-DMA destination buffer
- data1: Source data
- data2: Source data
- data3: Source data
- data4: Source data
- data5: Source data
- data6: Source data

Semantics:

- Check that the len parameter => 0 and <= 48, else return H_Parameter
- The destination LIOBN is checked for authorization per the remote triple of the one of the calling partition's **"ibm,my-dma-window"** property, else return H_D_Parm.
- The destination ioba and length are check for valid ranges per the remote triple of the one of the calling partition's **"ibm,my-dma-window"** property, else return H_D_Parm.
- Serialize access to the destination RTCE table with H_MIGRATE_DMA.
- The access bits of the associated RTCE table TCEs are checked for authorization, else return H_Permission.
- Copy len number of bytes from the data parameters starting at the high order byte of data1 toward the low order byte of data 6 into the buffer starting at the specified destination address, then return H_Success.

17.2.3.2.1.3 H_READ_RDMA

This hcall() copies up to 72 bytes of data from an RTCE table mapped buffer into a set of return registers.

Syntax:

```
int64 hcall(    /* H_Success, H_Parameter, H_S_Parm, H_Permission, H_Hardware */
      const uint64 H_READ_RDMA,       /* Performs Copy RDMA */
      int64     len,                  /* Length of transfer */
```

```
uint64    s-liobn,        /* LIOBN (RTCE table handle) of V-DMA source buffer */
uint64    s-ioba);        /* IO address of V-DMA source buffer */
```

Parameters:

- len: Length of transfer
- s-liobn: LIOBN (RTCE table handle) of V-DMA source buffer
- s-ioba: IO address of V-DMA source buffer

Semantics:

- Check that the len parameter => 0 and <= 72, else return H_Parameter
- The source LIOBN is checked for authorization per the remote triple of the one of the calling partition's **`"ibm,my-dma-window"`** property, else return H_S_Parm.
- The source ioba and length are check for valid ranges per the remote triple of the one of the calling partition's **`"ibm,my-dma-window"`** property, else return H_S_Parm.
- Serialize access to the source RTCE table with H_MIGRATE_DMA.
- The access bits of the associated RTCE table TCEs are checked for authorization, else return H_Permission.
- Copy len number of bytes from the source data buffer specified by s-liobn starting at s-ioba, into the registers R4 through R12 starting with the high order byte of R4 toward the low order byte of R12, then return H_Success.

17.2.3.2.2 Logical Remote DMA Option Requirements

R1–17.2.3.2.2–1. For the Logical Remote DMA option: The platform must implement the H_PUT_RTCE hcall() as specified in Section 17.2.2.2.1, "H_PUT_RTCE," on page 627.

R1–17.2.3.2.2–2. For the Logical Remote DMA option: The platform must implement the extensions to the H_PUT_TCE hcall() as specified in Section 17.2.2.2.7.1, "H_PUT_TCE, H_PUT_TCE_INDIRECT, and H_STUFF_TCE," on page 633.

R1–17.2.3.2.2–3. For the Logical Remote DMA option: The platform must implement the extensions to the H_MIGRATE_DMA hcall() as specified in Section 17.2.2.2.7.2, "H_MIGRATE_DMA," on page 634.

R1–17.2.3.2.2–4. For the Logical Remote DMA option: The platform must implement the H_COPY_RDMA hcall() as specified in Section 17.2.3.2.1.1, "H_COPY_RDMA," on page 643.

R1–17.2.3.2.2–5. For the Logical Remote DMA option: The platform must disable Logical Remote DMA operations that target an inactive partition (one that has terminated), including the H_COPY_RDMA hcall() and the H_PUT_RTCE hcall().

Implementation Note: It is expected that as part of meeting Requirement R1–17.2.3.2.2–5, all of the terminating partition's TCE table entries (regular and RTCE) are invalidated along with any clones (for information on invalidation of TCEs, see Section 17.2.2.2.4, "Redirected RDMA TCE Recovery and In-Flight DMA," on page 631). While other mechanisms are available for meeting this requirement in the case of H_COPY_RDMA, this is the only method for Redirected RDMA, and since it works in both cases, it is expected that implementations will use this single mechanism.

17.2.3.3 Subordinate CRQ Transport Option

For the synchronous infrastructure, in addition to the CRQ facility defined in Section 17.2.2.1, "Command/Response Queue (CRQ)," on page 621, the Subordinate CRQ Transport option may also be implemented in conjunction with the

CRQ facility. That is, the Subordinate CRQ Transport option requires that the Reliable Command/Response Transport option also be implemented. For this option, the Sub-CRQ facility defined in Section 17.2.2.3, “Subordinate Command/Response Queue (Sub-CRQ),” on page 634 is implemented.

17.2.3.3.1 Sub-CRQ Format and Registration

The format of the Sub-CRQ is as defined in Section 17.2.2.3, “Subordinate Command/Response Queue (Sub-CRQ),” on page 634.

The I/O address and length of the queue are registered using the H_REG_SUB_CRQ hcall(). See Section 17.2.3.3.5.1, “H_REG_SUB_CRQ,” on page 646.

17.2.3.3.2 Sub-CRQ Entry Format

See Section 17.2.2.3.2, “Sub-CRQ Entry Format,” on page 635.

17.2.3.3.3 Sub-CRQ Entry Processing

A sender uses the H_SEND_SUB_CRQ or H_SEND_SUB_CRQ_INDIRECT hcall() to enter one or more 32 byte messages on its partner’s queue. See Section 17.2.3.3.5.3, “H_SEND_SUB_CRQ,” on page 648 and Section 17.2.3.3.5.4, “H_SEND_SUB_CRQ_INDIRECT,” on page 649.

17.2.3.3.4 Sub-CRQ Transport Interrupt Notification

The receiver can enable and disable the virtual interrupt associated with its Sub-CRQ using the H_VIOCTL hcall(), with the appropriate subfunction. See Section 17.2.1.6, “H_VIOCTL,” on page 613. The interrupt number that is used in the H_VIOCTL call is obtained from the H_REG_SUB_CRQ call that is made to register the Sub-CRQ.

17.2.3.3.5 Sub-CRQ Transport hcall()s

The H_REG_SUB_CRQ and H_FREE_SUB_CRQ hcall()s are used by both client and server virtual IOA device drivers. It is the architectural intent that the hypervisor maintains a connection control structure for each defined partner/server connection. The H_REG_SUB_CRQ and its corresponding H_FREE_SUB_CRQ register and deregister partition resources with that connection control structure. However, there are several conditions that can arise architecturally with this connection process (the design of an implementation may preclude some of these conditions).

- The association connection to the partner virtual IOA not being defined (H_Not_Found).
- The partner virtual IOA CRQ connection may not have been completed (H_Closed).
- The partner may deregister its CRQ which also deregisters any associated Sub-CRQs.

17.2.3.3.5.1 H_REG_SUB_CRQ

This hcall() registers the RTCE table mapped memory that contains the Sub-CRQ. Multiple Sub-CRQ registrations may be attempted for each virtual IOA. If resources are not available to establish a Sub-CRQ, the H_REG_SUB_CRQ call will fail with H_Resource.

Programming Note: On platforms that implement the partition migration option, after partition migration the support for this hcall() might change, and the caller should be prepared to receive an H_Function return code indicating the platform does not implement this hcall(). If a virtual IOA exists in the device tree after migration that requires by

this architecture the presence of this hcall(), then if that virtual IOA exists after the migration, it can be expected that the hcall() will, also.

Syntax:

```
int64                                   /*H_Success: Registration completed*/
                                        /*H_Parameter: Failed due to Invalid Parameter */
                                        /*H_Not_Found: Failed due to undefined partner connection.*/
                                        /*H_Closed: Registration completed partner (CRQ) not connected.
                                        /*H_Resource: Failed due to lack of resources.
                                        /*H_Hardware: Function failed due to hardware error */
    hcall (const int64 H_REG_SUB_CRQ, /* Returns in R4 a cookie representing the Sub-CRQ number */
                                        /* Returns in R5 the interrupt number */
            uint64 unit-address,        /* As specified in the Virtual IOA's device tree node */
            uint64 Sub-CRQ-ioba,        /* I/O address of a Sub-CRQ (4 KB aligned)*/
            uint64 Sub-CRQ-length)      /* Length of the Sub-CRQ (multiple of 4 KB) */
```

Parameters:

- unit-address: Unit Address per device tree node **`"reg"`** property.
- Sub-CRQ-ioba: I/O address (offset into the RTCE table, as specified by the first window pane of the virtual IOA's **`"ibm,my-dma-window"`** property) of the Sub-CRQ buffer (starting on a 4 KB boundary).
- Sub-CRQ-length: Length of the Sub-CRQ in bytes (a multiple of 4 KB).

Semantics:

- Validate unit-address, else H_Parameter.
- Validate Sub-CRQ-ioba, which is the I/O address of the Sub-CRQ (I/O addresses for entire buffer length starting at the specified I/O address are translated by the RTCE table, is 4 KB aligned, and length, Sub-CRQ-length, is a multiple of 4 KB), else H_Parameter.
- Validate that there are sufficient resources associated with the Unit Address to allocate the Sub-CRQ, else H_Resource.
- Initialize the Sub-CRQ enqueue pointer and length variables. These variables are kept in terms of I/O addresses so that page migration works and any remapping of TCEs is effective.
- Initialize all Sub-CRQ entry header bytes to 0 (invalid).
- Disable Sub-CRQ interrupts.
- Place cookie representing Sub-CRQ number (will be used in H_SEND_SUB_CRQ, H_SEND_SUB_CRQ_INDIRECT, and H_FREE_SUB_CRQ) in R4.
- Place interrupt number (the same as will be returned by H_XIRR or H_IPOLL for the interrupt from this Sub-CRQ) in R5.
- If the CRQ connection is already complete, then return H_Success, else return H_Closed.

17.2.3.3.5.2 H_FREE_SUB_CRQ

This hcall() deregisters the RTCE table mapped memory that contains the Sub-CRQ. Note that the H_FREE_CRQ hcall() also deregisters any Sub-CRQs associated with the CRQ being deregistered by that hcall().

Programming Note: On platforms that implement the partition migration option, after partition migration the support for this hcall() might change, and the caller should be prepared to receive an H_Function return code indicating the

platform does not implement this hcall(). If a virtual IOA exists in the device tree after migration that requires by this architecture the presence of this hcall(), then if that virtual IOA exists after the migration, it can be expected that the hcall() will, also.

Syntax:

```
int64                                   /* H_Success, H_Parameter, H_Hardware, H_Busy, */
                                        /* H_LongBusyOrder1mSec, H_LongBusyOrder10mSec */
    hcall (const int64 H_FREE_SUB_CRQ, /* Function Code */
            uint64 unit-address,        /* As specified in the Virtual IOA's device tree node */
            uint64 Sub-CRQ-num)         /* The Sub-CRQ # cookie returned from H_REG_SUB_CRQ */
```

Parameters:

- unit-address: Unit Address per device tree node **"reg"** property.
- Sub-CRQ-num: The queue # cookie returned from H_REG_SUB_CRQ hcall() at queue registration time.

Semantics:

- Validate unit-address and Sub-CRQ-num, else H_Parameter
- Mark the connection to the associated partner partition as closed for the specified Sub-CRQ (so that send hcall()s from the partner partition fail).
- Mark the Sub-CRQ enqueue pointer and length variables for the specified Sub-CRQ as invalid.
- Disable Sub-CRQ interrupts for the specified Sub-CRQ.
- Return H_Success.

17.2.3.3.5.3 H_SEND_SUB_CRQ

This hcall() sends one 32 byte entry to the partner partition's registered Sub-CRQ.

Programming Note: On platforms that implement the partition migration option, after partition migration the support for this hcall() might change, and the caller should be prepared to receive an H_Function return code indicating the platform does not implement this hcall(). If a virtual IOA exists in the device tree after migration that requires by this architecture the presence of this hcall(), then if that virtual IOA exists after the migration, it can be expected that the hcall() will, also.

Syntax:

```
int64                                   /* H_Success, H_Parameter, H_Dropped, H_Hardware, H_Closed */
    hcall (const int64 H_SEND_SUB_CRQ, /* Function Code */
            uint64 unit-address,        /* As specified in the Virtual IOA's device tree node */
            uint64 Sub-CRQ-num          /* The Sub-CRQ# cookie from H_REG_SUB_CRQ (from partner)*/
            uint64 msg-dword0,          /* High order 8 bytes of message starting with the header byte */
            uint64 msg-dword1,          /* Next 8 bytes of message */
            uint64 msg-dword2,          /* Next 8 bytes of message */
            uint64 msg-dword3)          /* Low order 8 bytes of message */
```

Parameters:

- unit-addr: Unit Address per device tree node **"reg"** property.
- Sub-CRQ-num: The queue # cookie returned from H_REG_SUB_CRQ hcall() at queue registration time.
- msg-dword0: firmware checks only high order byte.

- msg-dword1, msg-dword2, msg-dword3: the rest of the message; firmware does not validate.

Semantics:

- Validate the Unit Address, else return H_Parameter.
- Validate that the Sub-CRQ, as specified by Sub-CRQ-num, is properly registered by the partner, else return H_Parameter.
- Validate that the message header byte (high order byte of msg-dword0) is 0x80, else return H_Parameter.
- Validate that there is an authorized CRQ connection to another partition associated with the Unit Address and that the associated CRQ is enabled, else return H_Closed.
- Enter Critical Section on target Sub-CRQ.
  - Validate that there is room on the specified Sub-CRQ for the message and allocate that message, else exit critical Section and return H_Dropped.
  - Store msgdword1 into bytes 4-7 of the allocated queue element.
  - Store msgdword2 into bytes 8-11 of the allocated queue element.
  - Store msgdword3 into bytes 12-15 of the allocated queue element.
  - Store order barrier.
  - Store msgdword0 into bytes 0-3 of the allocated queue element (this sets the valid bit in the header byte).
- Exit Critical Section.
- If receiver queue interrupt mode is enabled, then signal interrupt.
- Return H_Success.

17.2.3.3.5.4 H_SEND_SUB_CRQ_INDIRECT

This hcall() sends one or more 32 byte entries to the partner partition's registered Sub-CRQ. On H_Success, all of the entries have been put onto the Sub-CRQ. On any return code other than H_Success, none of the entries have been put onto the Sub-CRQ.

Programming Note: On platforms that implement the partition migration option, after partition migration the support for this hcall() might change, and the caller should be prepared to receive an H_Function return code indicating the platform does not implement this hcall(). If a virtual IOA exists in the device tree after migration that requires by this architecture the presence of this hcall(), then if that virtual IOA exists after the migration, it can be expected that the hcall() will, also.

Syntax:

```
int64                                   /* H_Success, H_Parameter, H_Dropped, H_Hardware, H_Closed */
    hcall (const int64 H_SEND_SUB_CRQ, /* Function Code */
            uint64 unit-address,        /* As specified in the Virtual IOA's device tree node */
            uint64 Sub-CRQ-num          /* The Sub-CRQ # cookie from H_REG_SUB_CRQ (from partner)*/
            uint64 ioba,                /* Address of buffer containing queue entries to be sent*/
            uint64 num-entries)         /* Number of queue entries in the buffer to be sent*/
```

Parameters:

- unit-addr: Unit Address per device tree node **"reg"** property.

- Sub-CRQ-num: The Sub-CRQ # cookie returned from H_REG_SUB_CRQ hcall() at queue registration time.
- ioba: The address of the TCE-mapped page which contains the entries to be placed onto the specified Sub-CRQ.
- num-entries: Number of entries to be placed onto the specified Sub-CRQ from the TCE mapped page starting at ioba (maximum number of entries is 16 in order to minimize the hcall() time).

Semantics:

- Validate the Unit Address, else return H_Parameter.
- Validate that the Sub-CRQ, as specified by Sub-CRQ-num, is properly registered by the partner, else return H_Parameter.
- If ioba is outside of the range of the calling partition assigned values, then return H_Parameter.
- If num-entries is not in the range of 1 to 16, then return H_Parameter.
- Validate that there is an authorized CRQ connection to another partition associated with the Unit Address and that the associated CRQ is enabled, else return H_Closed.
- Copy (num-entries * 32) bytes from the page specified starting at ioba to a temporary hypervisor page for contents verification and processing (this avoids the problem of the caller changing call by reference values after they are checked).
- Validate that the message header bytes for num-entries starting at ioba are 0x80, else return H_Parameter.
- Enter Critical Section on target Sub-CRQ.
  - Validate that there is room on the specified Sub-CRQ for num-entries messages and allocate those messages, else exit critical Section and return H_Dropped.
  - For each of the num-entries starting at ioba
    - Store entry bytes 1-31 into bytes 1-31 of the allocated queue element.
    - Store order barrier.
    - Store entry byte 0 into bytes 0 of the allocated queue element (this sets the valid bit in the header byte).
  - Loop
- Exit Critical Section.
- If receiver queue interrupt mode is enabled, then signal interrupt.
- Return H_Success.

17.2.3.3.6 Subordinate CRQ Transport Option Requirements

R1–17.2.3.3.6–1. For the Subordinate CRQ Transport option: The platform must implement the Reliable Command/Response Transport option, as defined in Section 17.2.3.1, “Reliable Command/Response Transport Option,” on page 637.

R1–17.2.3.3.6–2. For the Subordinate CRQ Transport option: The platform must implement the Sub-CRQ facility, as defined in Section 17.2.2.3, “Subordinate Command/Response Queue (Sub-CRQ),” on page 634.

R1–17.2.3.3.6–3. For the Subordinate CRQ Transport option: The platform must implement the H_REG_SUB_CRQ hcall(). See Section 17.2.3.3.5.1, “H_REG_SUB_CRQ,” on page 646.

R1–17.2.3.3.6–4. For the Subordinate CRQ Transport option: The platform must implement the H_FREE_SUB_CRQ hcall(). See Section 17.2.3.3.5.2, “H_FREE_SUB_CRQ,” on page 647.

R1–17.2.3.3.6–5. For the Subordinate CRQ Transport option: The platform must implement the H_SEND_SUB_CRQ hcall(). See Section 17.2.3.3.5.3, “H_SEND_SUB_CRQ,” on page 648.

R1–17.2.3.3.6–6. For the Subordinate CRQ Transport option: The platform must implement the H_SEND_SUB_CRQ_INDIRECT hcall(). See Section 17.2.3.3.5.4, “H_SEND_SUB_CRQ_INDIRECT,” on page 649.

R1–17.2.3.3.6–7. For the Subordinate CRQ Transport option: The platform must implement all of the following subfunctions of the H_VIOCTL hcall() (See Section 17.2.1.6, “H_VIOCTL,” on page 613):

- DISABLE_ALL_VIO_INTERRUPTS
- DISABLE_VIO_INTERRUPT
- ENABLE_VIO_INTERRUPT

17.3 Virtual Network Interface Controller (VNIC)

This section defines a Virtual Network Interface Controller (VNIC) interface to a server partition interfacing to a physical Network Interface Controller (NIC) adapter that allows multiple partitions to share a physical port. The implementation support is provided by code running in a server partition that uses the mechanisms of the Synchronous VIO Infrastructure (or equivalent thereof as seen by the client) to service I/O requests for code running in a client partition[1]. The client partition appears to enjoy the services of its own NIC adapter. The terms server and client partitions refer to platform partitions that are respectively servers and clients of requests, usually I/O operations, using the physical NIC that is assigned to the server partition. This allows a platform to have more client partitions than it may have physical NICs because the client partitions share I/O adapters via the server partition.

The VNIC model makes use of Remote DMA which is built upon the architecture specified in the following sections:

- Section 17.2.1, “VIO Infrastructure - General,” on page 600
- Section 17.2.2, “Partition Managed Class Infrastructure - General,” on page 620
- Section 17.2.3, “Partition Managed Class - Synchronous Infrastructure,” on page 637

The use of Remote DMA has implications that the physical NIC be able to do some of its own vitualization. For example, for an Ethernet adapter, being able to route receive requests, via DMA to the appropriate client partition, based on the addressing in the incoming packet.

17.3.1 VNIC General

This section contains an informative outline of the architectural intent of the use of VNIC. Other implementations of the server and client partition code, consistent with this architecture, are possible and may be preferable.

The client partition provides the virtual equivalent of a single port NIC adapter via each VNIC client IOA. The platform, through the partition definition, provides means for defining the set of virtual IOA’s owned by each partition and their respective location codes. The platform also provides, through partition definition, instructions to connect each client partition’s VNIC client IOA to a specific server partition’s VNIC server IOA. The mechanism for specifying this partition definition is beyond the scope of this architecture. The human readable handle associated with the partition definition of virtual IOAs and their associated interconnection and resource configuration is the virtual location code. The OF unit address (unit ID) remains the invariant handle upon which the OS builds its “physical to logical” configuration. The platform also provides a method to assign unique MAC addresses for each VNIC client adapter. The mechanism for allocating port names is beyond the scope of this architecture.

The client partition’s device tree contains one or more nodes notifying the partition that it has been assigned one or more virtual adapters. The node’s **“type”** and **“compatible”** properties notify the partition that the virtual adapter is a VNIC. The unit address of the node is used by the client partition to map the virtual device(s) to the OS’s corresponding logical representations. The **“ibm,my-dma-window”** property communicates the size of the RTCE table window panes that the hypervisor has allocated. The node also specifies the interrupt source number that has been assigned to the Reliable Command/Response Transport connection and the RTCE range that the client partition device driver may use to map its memory for access by the server partition via Logical Remote DMA. The client partition, uses the hcall()s associated with the Reliable Command/Response Transport facility to register and deregister its CRQ, manage notification of responses, and send command requests to the server partition. The client partition uses the hcall()s associated with the Subordinate CRQ Transport facility to register and deregister any sub-CRQs necessary for the operations of the VNIC.

1.The server partition for VNIC is expected to be part of the platform firmware and therefore an OS interface is not provided in this architecture for the server side. However, the platform firmware is still expected to be implemented by a partition, and hence the term “server partition.”

The client partition, upon noting the device tree entry for the virtual adapter, loads the device driver associated with the value of the **"compatible"** property. The device driver, when configured and opened, allocates memory for its CRQ (an array, large enough for all possible responses, of 16 byte elements), pins the queue and maps it into the I/O space of the RTCE window specified in the **"ibm,my-dma-window"** property using the standard kernel mapping services that subsequently use the H_PUT_TCE hcall(). The queue is then registered using the H_REG_CRQ hcall(). Next, I/O request control blocks (within which the I/O requests commands are built) are allocated, pinned, and mapped into I/O address space. Finally, the device driver registers to receive control when the interrupt source specified in the virtual IOA's device tree node signals.

Once the CRQ is setup, the device driver in the client queues an Initialization Command/Response with the second byte of "Initialize" in order to attempt to tell the hosting side that everything is setup on the hosted side. The response to this send may be that the send has been dropped or has successfully been sent. If successful, the sender should expect back an Initialization Command/Response with a second byte of "Initialization Complete," at which time the communication path can be deemed to be open. If dropped, then the sender waits for the receipt of an Initialization Command/Response with a second byte of "Initialize," at which time an "Initialization Complete" message is sent, and if that message is sent successfully, then the communication path can be deemed to be open.

Once the CRQ connection is complete between the client and the server, the client receives from the server the number of sub-CRQs that can be supported on the client side. The client allocates memory for the first sub-CRQ (an array, large enough for all possible responses, of 32 byte elements), pins the queue and maps it into the I/O space of the RTCE window specified in the **"ibm,my-dma-window"** property using the standard kernel mapping services that subsequently use the H_PUT_TCE hcall(). The queue is then registered using the H_REG_SUB_CRQ hcall(). This process continues until all desired sub-CRQs are registered or until the H_REG_SUB_CRQ hcall() indicates that the resources allocated to the client for sub-CRQs for the virtual IOA have already been allocated (H_Resource returned). Interrupt numbers for the Sub-CRQs that have been registered, are returned by the H_REG_SUB_CRQ hcall() (See Section 17.2.3.3.5.1, "H_REG_SUB_CRQ," on page 646).

Once all the CRQs and Sub-CRQs are setup, the communications between the client and server device drivers may commence for purposes of further setup operations, and then normal I/O requests, error communications, etc. The protocol for this communications is beyond the scope of this architecture.

17.3.2 VNIC Requirements

This normative section provides the general requirements for the support of VNIC.

R1–17.3.2–1. For the VNIC option: The platform must implement the Reliable Command/Response Transport option as defined in Section 17.2.3.1, "Reliable Command/Response Transport Option," on page 637.

R1–17.3.2–2. For the VNIC option: The platform must implement the Subordinate CRQ Transport option as defined in Section 17.2.3.3, "Subordinate CRQ Transport Option," on page 645.

R1–17.3.2–3. For the VNIC option: The platform must implement the Logical Remote DMA option as defined in Section 17.2.3.2, "Logical Remote DMA (LRDMA) Option," on page 642.

R1–17.3.2–4. For the VNIC option: The platform's OF device tree for client partitions must include as a child of the **/vdevice** node, at least one node of name **"vnic"**.

R1–17.3.2–5. For the VNIC option: The platform's **vnic** OF node must contain properties as defined in Table 236, "Properties of the vnic Node in the OF Device Tree," on page 654 (other standard I/O adapter properties are permissible as appropriate).

Table 236. Properties of the **vnic** Node in the OF Device Tree

Property Name	Required?	Definition
"name"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the virtual device name, the value shall be "vnic".
"device_type"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the virtual device type, the value shall be "network".
"model"	NA	Property not present.
"compatible"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the programming models that are compatible with this virtual IOA, the value shall include "IBM,vnic".
"used-by-rtas"	See definition column	Present if appropriate.
"ibm,loc-code"	Y	Property name specifying the unique and persistent location code associated with this virtual IOA.
"reg"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the unit address (unit ID) associated with this virtual IOA presented as an encoded array as with **encode-phys** of length **"#address-cells"** value shall be 0xwhatever (virtual **"reg"** property used for unit address no actual locations used, therefore, the size field has zero cells (does not exist) as determined by the value of the **"#size-cells"** property).
"ibm,my-dma-window"	Y	Property name specifying the DMA window associated with this virtual IOA presented as an encoded array of three values (LIOBN, phys, size) encoded as with **encode-int, encode-phys,** and **encode-int.**
"interrupts"	Y	Standard property name specifying the interrupt source number and sense code associated with this virtual IOA presented as an encoded array of two cells encoded as with **encode-int** with the first cell containing the interrupt source number, and the second cell containing the sense code 0 indicating positive edge triggered. The interrupt source number being the value returned by the H_XIRR or H_IPOLL hcall().
"ibm,my-drc-index"	For DR	Present if the platform implements DR for this node.
"ibm,#dma-size-cells"	See definition column	Property name, to define the package's dma address size format. The property value specifies the number of cells that are used to encode the size field of dma-window properties. This property is present when the dma address size format cannot be derived using the method described in the definition for the **"ibm,#dma-size-cells"** property in Appendix B, "LoPAPR Binding," on page 661.
"ibm,#dma-address-cells"	See definition column	Property name, to define the package's dma address format. The property value specifies the number of cells that are used to encode the physical address field of dma-window properties. This property is present when the dma address format cannot be derived using the method described in the definition for the **"ibm,#dma-address-cells"** property in Appendix B, "LoPAPR Binding," on page 661.
"local-mac-address"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the locally administered MAC addresses are denoted by having the low order two bits of the high order byte being 0b10.
"mac-address"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], specifying the initial MAC address (may be changed by a VNIC CRQ command).
"supported-network-types"	Y	Standard property name as per *Open Firmware Recommended Practice: Device Support Extensions* [5]. Reports possible types of "network" the device can support.
"chosen-network-type"	Y	Standard property name as per *Open Firmware Recommended Practice: Device Support Extensions* [5]. Reports the type of "network" this device is supporting.

Table 236. Properties of the **vnic** Node in the OF Device Tree ***(Continued)***

Property Name	Required?	Definition
"max-frame-size"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], to indicate maximum packet size.
"address-bits"	Y	Standard property name per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], to indicate network address length.
"interrupt-ranges"	Y	Standard property name that defines the interrupt number(s) and range(s) handled by this device. Subordinate CRQs associated with this VNIC use interrupt numbers from these ranges.

A SPLPAR Characteristics Definitions

This appendix defines the string that is returned by the *ibm,get-system-parameter* RTAS call when the parameter token value of 20 (SPLPAR Characteristics) is specified on the *ibm,get-system-parameter* RTAS call as per Section 7.3.16, “System Parameters Option,” on page 207**.**

A.1 SPLPAR Terms

The LoPAPR Shared Processor Logical Partition option (SPLPAR) defines many terms as presented in Table 237, “SPLPAR Terms,” on page 657.

Table 237. SPLPAR Terms

Term	Definition
Bound Threads	For virtual processor dispatches, if the hypervisor always dispatches a set of virtual threads together on a physical processor, the threads are said to be bound. This allows an operating system to make scheduling decisions based on cache affinity and work load. A change in this characteristic takes effect on the next reboot of the partition.
Capacity Increment	This defines the delta by which the entitled capacity of a partition can be incremented or decremented by DLPAR/WLM. The capacity increment is expressed as a percentage of a physical processor. A change in the capacity increment takes effect on the next reboot of the partition.
Desired Entitled Capacity	The desired entitled capacity set by the system administrator in the partition definition. The desired entitled capacity is expressed as a percentage of a physical processor. The desired entitled capacity can change without a reboot of the partition. The entitled capacity that the partition is currently using may differ from the desired entitled capacity because of WLM actions or failed system processors.
Desired Memory	The desired memory set by the system administrator in the partition definition. The desired memory is expressed in MB of storage. The desired memory can change without a reboot of the partition. The desired memory that the partition is currently using may differ from the desired memory because of WLM actions or because of failed system memory.
Desired Number of Processors	The desired number of virtual processors set by the system administrator in the partition definition. The desired number of processors can change without a reboot of the partition. The number of processors that the partition is currently using may differ from the desired number of processors because of WLM actions or because of failed system processors.
Desired Variable Capacity Weight	The desired variable capacity weight set by the system administrator in the partition definition. The desired variable capacity weight is a number between 0 and 255. The desired variable capacity weight can change without a reboot of the partition. The variable capacity weight that the partition is currently using may differ from the desired variable capacity because of WLM actions.
Dispatch Wheel Rotation Period	The duration of the hypervisor’s scheduling window. The time over which the entitled capacity of a virtual processor has to be utilized by the partition. At the start of a dispatch wheel rotation period, each virtual processor is eligible for CPU time corresponding to its entitled capacity. If the entire entitled capacity of a virtual processor is not utilized during a dispatch wheel rotation period, the unused entitled capacity is lost. The dispatch wheel rotation period is expressed as N number of time base ticks. The dispatch wheel duration of a partition with a capacity increment of 100 is 0. A change in the dispatch wheel rotation period takes effect on the next reboot of the partition.

Table 237. SPLPAR Terms

Term	Definition
Minimum Entitled Capacity	The minimum entitled capacity that is needed to power on the partition. The capacity is expressed as a percentage of a physical processor. The minimum entitled capacity is set by the system administrator in the partition definition. DLPAR cannot take the entitled capacity below the minimum entitled capacity. A change in the minimum entitled capacity takes effect on the next reboot of the partition. A partition can give up its entitled capacity to be below the minimum entitled capacity.
Minimum Entitled Capacity per Virtual Processor	The minimum entitled capacity that the platform requires for a virtual processor of any partition on the platform. The minimum virtual per virtual processor is enforced by the HMC in the partition definition and by the hypervisor for H_SET_PPP (Section 14.11.3.7, “H_SET_PPP,” on page 464). A change in the minimum entitled capacity per virtual processor takes effect on the next reboot of the partition.
Minimum Memory	The minimum amount of main store that is needed to power on the partition. Minimum memory is expressed in MB of storage. The minimum memory is set by the system administrator in the partition definition. DLPAR cannot take the partition memory below the minimum memory. A change in the minimum memory takes effect on the next reboot of the partition. A partition can always give up its memory to go below the minimum memory.
Minimum Number of Processors	The minimum number of virtual processors that are needed to start the partition. The minimum number of virtual processors is set by the system administrator in the partition definition. DLPAR cannot take the number of virtual processors below the minimum number of processors. A change in the minimum number of processors takes effect on the next reboot of the partition. A partition can always give up its virtual processors to go below the minimum number of processors.
Maximum Entitled Capacity	The maximum entitled capacity currently that can be assigned to the partition through DLPAR/WLM. The capacity is expressed as a percentage of a physical processor. The Maximum entitled capacity is set up by the system administrator in the partition definition. A change in the maximum entitled capacity maximum takes effect on the next reboot of the partition.
Maximum Platform Processors	The maximum number of processors that can be active on the platform. A change in the maximum platform processors takes effect on the next reboot of the partition.
Dedicated Donate Mode	For a partition with a capacity increment of 100, the platform uses a dedicated CPU to actualize a virtual processor of the partition. For such a partition, the platform can increase the capacity of the shared processor pool by utilizing the unused processor capacity of the partition. If the platform supports the dedicated donate function, it can be enabled by the system administrator in the partition definition. The value of this characteristic can change without a reboot of the partition.
Thread	A multi threaded processor may have multiple contexts executing concurrently. Each of them is called a thread. From a software viewpoint, a thread is an independent executing unit. Threads on a processor share some of the architected and unarchitected resources of a physical processor.

A.2 Key Words And Values

Table Table 238, “SPLPAR Characteristics,” on page 658 defines the key words and the associated legal values that will be returned in the ASCII NULL terminated string when the parameter token value of 20 (SPLPAR characteristics) is specified on the *ibm,get-system-parameter* RTAS call. The key word and value is separated by an ascii equal (“=”). Each key word, value pair is delimited by an ascii comma in the string. The numerical value of the characteristic corresponding to the key word is the decimal number that corresponds to the numeric characters in the value part of the key word, value pair.

Table 238. SPLPAR Characteristics

Characteristics	Key Word	Values	Notes
Bound Threads	BoundThrds	0,1	1
Capacity Increment	CapInc	1 through 100	
Dispatch Wheel Rotation Period	DisWheRotPer	1 through N	

Table 238. SPLPAR Characteristics

Characteristics	Key Word	Values	Notes
Minimum Entitled Capacity	MinEntCap	0 through N	2
Minimum Entitled Capacity per Virtual Processor	MinEntCapPerVP	1 through 100	
Minimum Memory	MinMem	0 through N	
Minimum Number of Processors	MinProcs	0 through N	
Maximum Entitled Capacity	MaxEntCap	1 through N	3
Maximum Platform Processors	MaxPlatProcs	1 through N	
Desired Entitled Capacity	DesEntCap	0 through N	4
Desired Memory	DesMem	0 through N	5
Desired Number of Processors	DesProcs	0 through N	6
Desired Variable Capacity Weight	DesVarCapWt	0 through 255	
Dedicated Donate Mode	DedDonMode	0,1	7

Notes:

1. 0=Threads are not bound, 1=Threads are bound.
2. The maximum numeric value of Minimum Entitled Capacity is the number of processors on the platform multiplied by 100.
3. The maximum numeric value of Maximum Entitled Capacity is the number of processors on the platform multiplied by 100.
4. The numeric value of Desired Entitled Capacity is greater or equal than the numeric value of the Minimum Entitled Capacity and less than or equal to the numeric value of the Maximum Entitled Capacity.
5. The numeric value of Desired Memory is greater or equal than the numeric value of the Minimum Memory and less than or equal to the maximum memory set by the system administrator in the partition profile.
6. The numeric value of Desired Number of Processors is greater or equal than the numeric value of the Minimum Number of Processors and less than or equal to the maximum number of virtual processors set by the system administrator in the partition profile.
7. 0=Dedicated Donate Mode is disabled, 1=Dedicated Donate Mode is enabled.

B LoPAPR Binding

B.1 Purpose of this System Binding

This appendix specifies the application of OF to an LoPAPR System, including requirements and practices to support unique hardware and firmware specific to the platform implementation. The core requirements and practices specified by OF must be augmented by system-specific requirements to form a complete specification for the firmware implementation of an LoPAPR System. This appendix establishes such additional requirements pertaining to the platform and the support required by OF.

B.2 Overview

This appendix specifies the application of *IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware, Core Practices and Requirements*, *Core Errata, IEEE P1275.7* and appropriate OF Standards for LoPAPR computer systems, including practices for client program interface and data formats.

B.2.1 General Requirements for OF

An OF implementation for an LoPAPR platform shall implement the core requirements as defined in *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], core errata *Core Errata, IEEE P1275.7/D4* [3], the PA Processor-specific extensions described in Appendix C, "PA Processor Binding," on page 753, other appropriate bindings and/or recommended practices contained in the references (see "Bibliography" on page 889), and the LoPAPR Binding specific extensions described in this appendix.

In addition, an OF implementation for an LoPAPR platform shall implement the *Device Interface*, *Client Interface* and *User Interface* as defined in *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2].

Some LoPAPR Binding property names exceed the OF Base specification limit of 31 characters. LoPAPR OF implementations shall support property names of at least 47 characters.

B.3 Terms

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

ARP — Address Resolution Protocol

BOOTP — Bootstrap Protocol

CHRP — Common Hardware Reference Platform

core, core specification, core document

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

core errata	Refers to *Core Errata*, *IEEE P1275.7*
CPU	Central Processing Unit
ELF	Executable and Linking Format. A binary object file format defined by *System V Application Binary Interface, PowerPC Processor Supplement* [15] that is used to represent *client programs* in OF for PA.
FDISK	Refers to the boot-record and partition table format used by MS-DOS, as defined in *MS-DOS Programmer's Reference* [12].
gateway	Network connecting device
host	A computer. In particular a source or destination of messages from the point of view of the communication network.
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IP	Internet Protocol
IO	Input/Output
LAN	Local Area Network
NVRAM	Non-volatile memory that is the repository for various platform, OF and OS information that remains persistent across reboots, power management activities and/or cycles.
Open Firmware (OF)	The firmware architecture defined by *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2] and *Core Errata, IEEE P1275.7/D4* [3], or, when used as an adjective, a software component compliant with the core specification and errata.
PCU	Power Configuration Utility; Refers to a platform program to assist a user to manage device power.
PE	Portable Executable. A binary object file format defined by *Peering Inside the PE: A Tour of the Win32 Portable Executable File Format* [13].
PROM	programmable read only memory
real-mode	The mode in which OF and its client are running with translation disabled; all addresses passed between the client and OF are real (i.e., hardware) addresses.
RFC	Internet Request For Comments; part of the technical process of establishing a standard.
ROM	Read Only Memory
suspend	A form of Power Management characterized by a fast recovery to full operation. Typically, system memory will not be powered off while in the suspend state.
TFTP	Trivial File Transfer Protocol
UDP	User Datagram Protocol
virtual-mode	The mode in which OF and its client share a single virtual address space, and address translation is enabled; all addresses passed between the client and OF are virtual (translated) addresses.

B.4 LoPAPR Boot Flow

This section gives a system boot process overview and defines the enhancements to the standard OF boot process that are present in the boot process for an LoPAPR platform.

B.4.1 Boot Overview

The platform performs a normal OF boot (see *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], as stated in the Core Practice Document, Section 4.2.3, Start-up script evaluation. LoPAPR platforms provide an additional capability to assist the user in choosing which of several OSs to boot. A key sequence can be used to interrupt the normal boot flow and present the user with a *multiboot menu*, which can be either graphical or text-based at the discretion of the platform's firmware, from which the user can choose one of the installed or installable OSs. Presenting the user with this choice can also be made the default mode of operation at platform boot time, by means of the **auto-boot?** and **menu?** configuration variables.

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

CHRP Start-up Script Sequence Evaluation:

a) Power On Self Test (POST)

b) System Initialization

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

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

e) **install-console**

f) **banner**

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

g) Secondary Diagnostics and other system-dependent initialization

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

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

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

The boot flow described above occurs after all of the devices have been probed (i.e., by the execution of **probe-all**); see Section B.4.1.1, "Additional Requirements for probe-all Method," on page 663 additional requirements for **probe-all** method.

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

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

B.4.1.1 Additional Requirements for probe-all Method

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

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

B.4.1.2 LoPAPR Multiboot

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

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

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

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

B.4.1.3 Bootinfo Configuration Variables

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

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

B.4.1.4 Bootinfo Properties

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

"bootinfo" S

property name locates the node's *bootinfo object*

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

The presence of this property signifies that the device has an associated *bootinfo object*. The value is a text string such that when this device's node **open** method is called, the value of text string that is passed to the device's node **open** method is "my-args". When so opened, subsequent calls to the node's **"read"** method will yield the contents of the node's *bootinfo objec*t.

B.4.1.5 Standard Locations for *Bootinfo Objects*

The standard locations for *bootinfo objects* on various LoPAPR media and partition types is shown in the table below. An OS must put its *bootinfo object* in the standard location in order to guarantee interoperability with the LoPAPR *multiboot menu* mechanism.

Table 239. Standard Pathnames for *bootinfo.txt* File

Name	Device/Partition	Notes
Installation Media:		
Any block device:	*device*:*partition*,\ppc\bootinfo.txt	Any file system format
Tape:	*device*:0 (Note 1)	Presence of bootinfo.txt is optional
ROM:	*device*:bootinfo	bootinfo is the value of the **"bootinfo"** property in a **/rom** child node
Network:	Could specify bootinfo.txt or some other file from the Bootp server	Specifying bootinfo.txt from the Bootp server is optional

Note 1: If bootinfo.txt file is not present, file 0 should contain a program image file for a bootable tape.

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

ALIAS EXAMPLE:

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

bootinfo-nt-4.0=disk:\os\winnt\bootinfo.txt

NON-ALIAS EXAMPLE:

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

B.4.1.6 Bootinfo Objects

The information used by OF to display information in the *multiboot menu* and to locate and process an OS load image is contained within a sequence of text that is called a *bootinfo object*. The text comprising the bootinfo object uses SGML syntax, as defined in *ISO Standard 8879:1986, Information Processing -- Text and Office Systems -- Standard Generalized Markup Language (SGML)* [16], with tags identifying the subordinate elements.

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

<CHRP-BOOT>

<DESCRIPTION>

....

</DESCRIPTION>

<OS-NAME>

....

```
</OS-NAME>
<BOOT-SCRIPT>
        ....
</BOOT-SCRIPT>
<ICON
        SIZE=ww,hh
        COLOR-SPACE=r,g,b
        >
                <BITMAP>
                        hh hh hh hh . . .
                </BITMAP>
</ICON>
</CHRP-BOOT>
```

Notes:

1. If ‘SIZE’ is not present, assume default of 64,64. If ‘COLOR-SPACE’ is not present, assume default of 3,3,2.
2. Another <chrp-boot> tag sequence could define a different boot selection
3. LoPAPR platforms will recognize only the tags between the beginning <chrp-boot> tag until the end </chrp-boot> tag. If a tag is unrecognized, the material will be ignored until the end tag. Other non-<chrp-boot> tags may be supported in the future. These additional selections would also be presented to the user as boot options.

B.4.1.6.1 Bootinfo Entities

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

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

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

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

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

B.4.1.6.2 Bootinfo Character Sets

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

B.4.1.6.3 Element Tag Descriptions

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

B.4.1.6.4 CHRP-BOOT Element

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

B.4.1.6.5 OS-NAME element

This element contains the complete name of the OS.

B.4.1.6.6 BOOT-SCRIPT element

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

B.4.1.6.7 ICON element

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

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

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

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

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

B.4.1.6.7.1 BITMAP element

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

icon string example: **<icon size=64,64 color-space=3,3,2><bitmap>hh hh...**

hh[2]</bitmap></icon>

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

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

AIX Bootinfo Object Example:

<chrp-boot>

<description>AIX 4.2.D.0</description>

<os-name>AIX 4.2.D.0</os-name>

<boot-script>boot &device;:2</boot-script>

<icon size=64,64 color-space=3,3,2><bitmap>hh ... hh[1]</bitmap></icon>

</chrp-boot>

AIX Diagnostics Bootinfo Object Example:

<chrp-boot>

<description>AIX 4.2.D.0 Diagnostics</description>

<os-name>AIX 4.2.D.0 Diagnostics</os-name>

<boot-script>boot &device;:2 diag</boot-script>

<icon size=64,64 color-space=3,3,2><bitmap>hh ... hh[1]</bitmap></icon>

</chrp-boot>

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

B.4.1.7 Multiboot Menu

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

Multiboot Required Functions:

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

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

There shall be at least one key sequence to enter the multi-boot platform function for an LoPAPR platform.

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

B.4.2 Reboot-Command Variable Description

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

reboot-command(-- addr len)N

One time or temporary reboot command.

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

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

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

B.5 LoPAPR Processor

OF defines a minimum cell size of 32 bits; therefore, only one cell is necessary to represent addresses up to 4GB (32 bits). Two cells are necessary to represent addresses above 4GB and within 64 bits. Also, two cells are necessary to represent sizes greater than 4GB.

B.5.1 Processor Endian-ness Support

LoPAPR requires the use of PA processors that support Big-Endian storage format. LoPAPR allows for the use of PA processors that support Little-Endian storage format in addition to Big-Endian storage format.

B.5.2 Multi-Threading Support

The processors used in some platforms support multiple threads of execution. This processor model differs from Symmetric Multi-Processors in that the multiple threads of execution share the processor hardware to such an extent that operations on one thread can significantly affect the performance of another tread of the same processor. Therefore, the processor is represented with a single processor node having multiple interrupt server numbers. The OS is then free to start and stop multi-threading as the processing environment dictates. The client interface call **start-cpu**, operates on the full CPU as presented in the device tree, upon successful completion, the started CPU is running in single threaded mode, the active thread being the one associated with the first interrupt server number in the **"ibm,ppc-interrupt-server#s"** property. The client interface calls: **stop-self, idle-self, resume-cpu** are all defined to operate on the full CPU when called in single threaded mode, the behavior of these calls if called with multiple threads active is implementation dependent, it is suggested that the implementation deactivate all but one thread before performing the call's standard function.

B.6 OF Platform Extensions

This section defines OF properties, methods, device tree structure and Client Interface Service requirements for LoPAPR platforms.

The naming conventions for IBM unique OF properties and devices are as follows:

- Properties created for use only by IBM compatible implementations must have the string **"ibm,"** as a prefix to the property name.
- Property names prefixed with the string **"ibm,fw-"** are reserved for and must be controlled by the Firmware Area.
- An IBM property name which does not have the firmware or AIX prefix must be defined in this document unless documented elsewhere.
- The value of a device **"name"** whether reported through the compatible property or name property for a device implemented by IBM must contain the string **"IBM,"** as a prefix unless it conforms to a binding which specifies otherwise.

B.6.1 Properties for Dynamic Reconfiguration

The following standard properties are define for all dynamically reconfigurable platform nodes.

"ibm,drc-indexes"

property name denotes an integer index to be used to communicate to the firmware what connector is to be operated upon for the various RTAS calls used for DR.

prop-encoded-array: An integer encoded as with **encode-int**, followed by a list of integers also encoded as with **encode-int**.

For each DR connector, a unique integer index is provided which uniquely identifies the DR connector for purposes of the *ibm,configure-connector*, *set-indicator,* and *get-sensor* RTAS calls. The first element of the array is the number of connectors associated with the node. The second element of the array is the index which represents the first connector associated with the node, the third element the second connector, and so on until all of the node's DR connectors are specified.

"ibm,my-drc-index"

property name denotes an integer index (value of the entry in the **"ibm,drc-indexes"** property) for the connector between the node and the node's parent.

prop-encoded-array: An array of integers encoded as with **encode-int**.

"ibm,drc-names"

property name describes the external labeling of the DR connectors.

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

For each DR connector, a unique human-readable name for a connector. The first element of the array is the number of connectors associated with the node. The second element of the array is the human-readable name which represents the first connector associated with the node, the third element the second connector, and so on until all of the node's DR connectors are specified.

"ibm,drc-power-domains"

property name gives the power domain number for each connector associated with the node, which is the domain number to be used in the *set-power-level* RTAS call, if necessary.

prop-encoded-array: An integer encoded as with **`encode-int`**, followed by a list of integers also encoded as with **`encode-int`**.

For each DR connector, the power domain which will be controlled for DR operations (the power domain in which the DRC resides), and which will be used, if not -1, in the *set-power-level* RTAS call for the connector. The power domain number of -1 denotes a live-insertion power domain (in which case, the *set-power-level* RTAS call is not used). The first element of the array is the number of connectors associated with this node. The second element represents the domain number for the first connector. The element following this is the domain number for the second connector, and so on until all of the node's DR connectors are specified.

`"ibm,drc-types"`

property name, describes the type of each connector associated with the node, in a human-readable form.

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

The first element of the array is the number of connectors associated with this node. The second element of the array is the connector type of the first connector associated with the node, the third element the second connector, and so on until all the node's DR connectors are specified, and these elements will be one of the currently defined connector types specified in Table 240, "Currently Defined DR Connector Types," on page 671.

Table 240. Currently Defined DR Connector Types

Connector Type (character string)	Description
1	A 32-bit, 5 Volt conventional PCI slot which accommodates cards that operate up to 33 MHz Only.
2	A 32-bit, 5 Volt conventional PCI slot which accommodates cards that operate up to 33 MHz.
3	A 32-bit, 3.3 Volt conventional PCI slot which accommodates cards that operate up to 33 MHz Only.
4	A 64-bit, 5 Volt conventional PCI slot which accommodates cards that operate up to 33 MHz Only.
5	A 64-bit, 5 Volt conventional PCI slot which accommodates cards that operate up to 33 MHz.
6	A 64-bit, 3.3 Volt conventional PCI slot which accommodates cards that operate up to 33 MHz Only.
7	A 32-bit, 3.3 Volt conventional PCI slot which accommodates cards that operate up to 66 MHz. IOAs that operate up to 66 MHz will only operate at frequencies above 33 MHz if there are no 33 MHz IOAs on the same bus.
8	A 64-bit, 3.3 Volt conventional PCI slot which accommodates cards that operate up to 66 MHz. IOAs that operate up to 66 MHz will only operate at frequencies above 33 MHz if there are no 33 MHz IOAs on the same bus.
9	Reserved
10	Reserved
11	A 32-bit PCI-X capable slot which accommodates cards that operate up to 66 MHz
12	A 32-bit PCI-X capable slot which accommodates cards that operate up to 100 MHz
13	A 32-bit PCI-X capable slot which accommodates cards that operate up to 133 MHz
14	A 64-bit PCI-X capable slot which accommodates cards that operate up to 66 MHz

Table 240. Currently Defined DR Connector Types ***(Continued)***

Connector Type (character string)	Description
15	A 64-bit PCI-X capable slot which accommodates cards that operate up to 100 MHz
16	A 64-bit PCI-X capable slot which accommodates cards that operate up to 133 MHz
17	A 64-bit PCI-X capable slot which accommodates cards that operate up to 266 MHz
18	A 64-bit PCI-X capable slot which accommodates cards that operate up to 533 MHz
19	A PCI Express Rev 1 slot with 1x lanes.
20	A PCI Express Rev 1 slot with 2x lanes.
21	A PCI Express Rev 1 slot with 4x lanes.
22	A PCI Express Rev 1 slot with 8x lanes.
23	A PCI Express Rev 1 slot with 16x lanes.
24	A PCI Express Rev 1 slot with 32x lanes.
25	A PCI Express Rev 2 slot with 1x lanes.
26	A PCI Express Rev 2 slot with 2x lanes.
27	A PCI Express Rev 2 slot with 4x lanes.
28	A PCI Express Rev 2 slot with 8x lanes.
29	A PCI Express Rev 2 slot with 16x lanes.
30	A PCI Express Rev 2 slot with 32x lanes.
CPU	Logical CPU
MEM	Logical Memory Region
MEM-n (where n is a non-zero integer)	Extended Logical Memory Region(s). Used with the Reserved Memory option.
PHB	Logical PCI Host Bridge
SLOT	Logical I/O slot
PORT	Logical Port

“ibm,phandle”

property name, defines the phandle for the node.

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

B.6.2 OF Root Node

This section defines additional properties and methods associated with LoPAPR platforms that OSs expect to find in the root node. Unit addresses in an LoPAPR system are limited to 60 bits in length corresponding to the maximum real address supported by the POWER processor architecture. The unit address of all non-system nodes that are children of the root node shall have the same value each time the platform is booted; i.e., shall be invariant for each boot process.

Notes:

1. This requirement ensures that the PHB would have a stable unit address. Violations of this rule may require reinstallation of an OS.
2. The recommended practice is to generate a virtual unit address for PHB nodes. This is done by giving a zero length to its first reg property with an address that is selected such that it remains constant. In single bridge platforms, the value is chosen based upon historical precedent of the predecessor product. In multi-enclosure platforms, the virtual unit address is based upon the manufacturing serial number to insure uniqueness.

B.6.2.1 Root Node Properties

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

`"#address-cells"` S

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

`"#size-cells"` S

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

`"clock-frequency"` S

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

`"ibm,extended-clock-frequency"`

property name: Property that represents the primary system bus speed in hertz of this node. This property allows the encoding of multi-giga-hertz quantities.

prop-encoded-array: Consisting of two cells (freq-hi, freq-lo) each encoded as with **`encode-int`**, such that their combined value is (freq-hi || freq-lo).

`"system-id"` S

Standard *property name*, encoded as with **`encode-string`**, that contains the identification of the computer system (Reference the **`"name"`** property in *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2]). This string should be unique across all systems and all manufacturers. An example of an address of this form is "0nnnnnnmmmmmm" where nnnnnn is a sequence of 6 uppercase hexadecimal digits representing a 24-bit value that identifies manufacturer and mmmmmm is a sequence of 6 uppercase hexadecimal digits representing a 24-bit binary number assigned by the manufacturer to assure uniqueness.

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

"model" S

Standard *property name* that is a printable string identifying the manufacturer and model number of the platform.

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

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

"device_type" S

Standard *property name* that is a printable string identifying the platform as LoPAPR Compliant.

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

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

"ibm,lpar-capable"

property name indicates that the platform is capable of supporting logical partitioning and is only present on such systems. This property is, however, present even if the platform is not currently configured for logical partition operation.

prop-encoded-array: <NULL>

"ibm,converged-loc-codes"

property name indicates that the platform supports the "Converged Location Code" option. This property shall be present only on platforms that support the "Converged Location Code" option.

prop-encoded-array: <NULL>

"ibm,max-boot-devices"

property name indicates the maximum number of boot-device entries that the OF automatic boot code will process (entries after this number are ignored). Platforms that do not present this property default to process a maximum of 5 entries.

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

"ibm,aix-diagnostics"

property name indicates that the platform is capable running AIX diagnostics.

prop-encoded-array: <NULL>

"ibm,diagnostic-lic"

property name, presented to partitions authorized to perform diagnostic operations, that indicates that the platform is designed to use the specified license internal code package for diagnostic services.

prop-encoded-array: one or more encapsulated package handles encoded as with **encode-int**.

"ibm,io-server-lic"

property name indicates that the platform is designed to use the specified license internal code package for I/O services.

prop-encoded-array: one or more encapsulated package handles encoded as with **encode-int**.

"ibm,plat-res-int-priorities"

property name that designates to the client program that the platform has reserved one or more interrupt priorities for its own use.

prop-encoded-value: one or more (*interrupt priority, range*) pairs, where *interrupt priority* is a single cell hexidecimal number between 0x00 and 0xFF, and *range* is an integer encoded as with **encode-int** that represents the number of contiguous interrupt priorities that have been reserved by the platform for its internal use.

"ibm,eeh-default"

property name indicates the platform's default setting for the EEH option.

prop-encoded-array: An integer encoded as with **encode-int** that represents the platform's default setting for the EEH option. The defined states are:

0= The platform boots up with the EEH option disabled.

1= The platform boots up with the EEH option enabled.

"ibm,model-class"

property name to indicate the platform class.

prop-encoded-array: string encoded as defined in Table 241, "Example Encoding Strings," on page 675.

Table 241. Example Encoding Strings

Encoded String	Platform Class
C5	Blade/Entry
D5	Entry
E5	Entry
F5	Mid-range
G5	High-end
H5	High-end
P5	obsolete

"ibm,partition-no" **S**

property name to define the partition number of this particular logical partition as established by the Hardware Management Console.

prop-encoded-array: The logical partition number is a one cell integer encoded as with **encode-int**.

"ibm,partition-name" **S**

property name to define the partition name of this particular logical partition as established by the Hardware Management Console.

prop-encoded-array: A NULL terminated string.

"ibm,platform-hardware-notification"

property name indicating to the OS the presence of hardware for which the OS may need to take action. This property exists to notify the OS of hardware elements on the platform which may require special handling by the OS, such as in response to a hardware errata.

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

The first element represents the number of strings to follow in the property. Each string in the array names a hardware element that may require the OS to take specific action. The intention is that the string is to name the hardware element being reported. It is not the intention to define (or even hint at) the action that the OS must take. It is expected that some source outside this document will contain a cross reference between these strings and documentation such as hardware errata notes which define the action the OS must take.

If the **"ibm,platform-hardware-notification"** property is provided and a string begins with the <name> field of the **"name"** (see Appendix C, "PA Processor Binding," on page 753) property in the **CPU** nodes followed by an underscore, the characters following the underscore are a hexadecimal representation of the contents of a Processor Version Register that the platform may contain.

"ibm,fault-behavior"

property name to define the behavior of the Error Log indicator relative to FRU faults.

prop-encoded-array: An integer encoded as with encode-int that represents how the Error Log indicator should be handled when a FRU fault is detected.

Property non-existent -- The OS may set FRU Fault and Error Log indicators for all errors (those it detected and those that the platform reports to the OS).

Property exists with a value of 1 -- The OS only sets FRU Fault and Error Log indicators for errors it detects.

"ibm,fru-9006-deactivate"

property name to define whether or not the OS should deactivate 9006 indicators that it has activated.

prop-encoded-array: An integer encoded as with encode-int that represents how the OS should behave relative to FRU Fault indicator deactivation.

Property non-existent -- The OS is responsible for deactivating FRU level 9006 indicators that it has activated.

Property exists with a value of 1 -- The OS should not deactivate FRU level 9006 indicators that it has activated, but is allowed to do so (firmware does not block). The deactivation of the FRU level 9006 indicators is platform and service procedure dependent.

"compatible" **S**

Standard *property name* that conveys the platform architecture identifiers.

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

Specifies a list of platform architectures with which this platform is compatible. This is used by a client program when it is trying to determine the appropriate support for this platform. This property shall include the substring "LoPAPR-<LoPAPR version>-<Manufacturer>-<Manufacturer Version>" where <LoPAPR version> is the text (without blanks) after the word "Version" on the cover page of the LoPAPR specification that the platform adheres to, <Manufacturer> is a unique string identifying the manufacturer of the platform (see the OF standard description of the **"name"** property for suggestions), and <Manufacturer_Version> is defined by the manufacturer of the platform.

Note: In order to comply with the OF Standard description of the **"compatible"** property, implementations should place the "LoPAPR-<LoPAPR version>-<Manufacturer>-<Manufacturer Version>" substring after values that were present in the **"compatible"** property prior to the inclusion of the "LoPAPR-<LoPAPR version>-<Manufacturer>-<Manufacturer Version>" substring.

"ibm,max-vios-function-level"

property name to define the maximum vios function level that a client shall permit.

prop-encoded-array: An integer encoded as with **encode-int** that represents the maximum VIOS level that the client shall negotiate. See Appendix E, "A Protocol for VSCSI Communications," on page 795 for the definition of the values of this property.

"ibm,partition-performance-parameters-level"

property name to define the level of partition performance parameter reporting supported by the platform.

prop-encoded-array: An integer encoded as with encode-int that represents the level of partition performance parameter reporting supported by the platform (See Table 242, "Level of Partition Performance Parameter Reporting Supported," on page 677).

Table 242. Level of Partition Performance Parameter Reporting Supported

Partition Performance Parameter Level	Description
0	Base Level
1	Addition of Processor Virtualization Resource Allocations to H_GET_PPP and Virtualization Processor idle count to H_PIC

"ibm,preconfigure-usb-kvm"

property name the presence of which indicates that the platform requires the operating system to force configuration of the USB keyboard/mouse nodes during its configuration phase.

prop-encoded-array: <NULL>

This property, when present in the root node, indicates that the platform requires the operating system to force pre-configuration of USB keyboard/mouse nodes internally during its configuration phase. This property is presented only by platforms with a KVM switch that desire to force configuration by one or more target operating systems that do not fully support dynamic addition of USB keyboard and mouse unless the USB keyboard and mouse are actually seen during the operating system configuration phase, but may be present even if the KVM switch is not present when the device tree is inspected. Forced pre-configuration is needed since the operating system may not actually see the USB keyboard and mouse during its configuration phase due to the KVM switch that the platform uses only shows USB keyboard and mouse when those devices are actually switched to the appropriate KVM switch port.

"ibm,enable-ci64-capable"

property name to define the platform supports the **"ibm,enable-ci64"** method in the Client Interface.

prop-encoded-array: None, this is a name only property.

"ibm,migratable-partition"

property name indicating that the platform supports the potential migration of this partition.

prop-encoded-array: NULL

"ibm,extended-address" S

property name indicates this platform supports Peripheral Memory Spaces, Peripheral I/O Spaces, and SCA spaces above 4 GB.

prop-encoded-array: <none>

This property must be present.

"ibm,ignore-hp-po-fails-for-dlpar"

property name to define that the OS may ignore failures of Hot Plug power off and isolate operations during a DLPAR remove operation. See also Note 2 in Figure 12, "Dynamic Reconfiguration State Transition Diagrams," on page 359.

prop-encoded-array: None, this is a name only property.

"ibm,managed-address-types"

property name that conveys the platform's supported types of external addresses that are reprogrammable.

prop-encoded-array: The concatenation, with **encode+**, of an arbitrary number of text strings as described in Table 243, "Address types supported in "ibm,managed-address-types" property," on page 678, each encoded with **encode-string**.

Table 243. Address types supported in **"ibm,managed-address-types"** property

Text String	Description
ethernet_mac	Ethernet MAC address
ethernet_vlan	Ethernet VLAN ID (for default traffic)
san_wwn	Fibre Channel World Wide Name (covers both Port & Node names)
sas_wwid	SAS IOA's WWID value

"ibm,service-indicator-mode"

property name indicates in which service indicator mode the platform is operating.

prop-encoded-array: an integer encoded as with *encode-int* that represents the mode. Defined values are:

- 0 = Platform is operating in the Guiding Light mode.
- 1 = Platform is operating in the Lightpath mode.

Implementation Notes:

1. In the absence of this property, the determination of how the OS is to behave is made by the platform presenting or not presenting FRU Fault indicators to the OS see chapter16, "Service Indicators," on page 511. In the case where there are no FRUs owned by the partition, the OS will not observe any FRU Fault indicators assigned, even when the platform is operating in the Lightpath mode.
2. Presenting this property does not imply any relaxation of the requirements spe3cified in chapter16, "Service Indicators," on page 511.

B.6.2.2 Properties of the Children of Root

"ibm,9009-domain"

property name that defines the index for a 9009 reset component indicator, and if it exists, the corresponding 9009 sensor, for the node in which the property exists. Multiple nodes may have the same index, indicating that they belong to the same reset domain; including nodes which are not descendents of the node which contains this property. Descendents of a node containing this property will be in the same reset domain.

prop-encoded-array: An integer encoded as with **encode-phys** that represents the index for the indicator, and if it exists, for the corresponding sensor.

"ibm,associativity"

property name to define the associativity domains for this resource.

prop-encoded-array: One or more associativity lists. Each associativity list consisting of a number of entries integer (N) encoded as with **encode-int** followed by N integers encoded as with **encode-int** each representing an associativity domain number.

B.6.2.3 Root Node Methods

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

Boot Loader Note: The suggested behavior for boot loader client programs:
1) Check the "ibm,rpa-client-config" property to see if the platform recognized the "ignore-my-settings" bit in the boot loader image i.e. YABOOT for LINUX.
2) If recognized, check for existence of "ibm,client-architecture-support" and invoke that method with the **ibm,**

compatible with the Real Base and Real Size constraints of the kernel being loaded.
3) If that method did not exist, invoke "PROCESS-ELF-HEADER" from /packages/elf-loader with a simulated ELF-header that the Linux kernel is compatible with.

ibm,client-architecture-support (ibm,architecture.vec -- err?)

This method is called via the call-method *Client Interface Service*, prior to starting other partition processors or threads, to communicate to the platform, via the **ibm,architecture.vec** structure, the architecture options that are supported by the client program. Based upon this knowledge the platform configures itself and the device tree to represent the most functional programming environment supported by the combination of the platform, client program and user specified constraints. If multiple partition processors or threads are active at the time of the **ibm,client-architecture-support** method call, or an error is detected in the format of the **ibm,architecture.vec** structure, the **err?** boolean shall be **TRUE;** else **FALSE**. The **ibm,architecture.vec** input parameter is the starting address of a self defining structure in contiguous memory. Some bits within the **ibm,architecture.vec** structure option vectors represent policies. When set, and an associated condition is detected, the **ibm,client-architecture-support** method does not return and processing continues as with a boot failure of the client program. The LoPAPR architecture options that are selected by this method are communicated in the value of the **"ibm,architecture-vec-5"** property of the **/chosen** node.

To ensure the greatest level of interoperability, the client program should constrain itself to using the set of instructions and environment specified for first level interrupt handlers, see Book III of the *Power ISA* [1], while not attempting access to potentially optional SPRs or the MSR prior to invoking the **ibm,client-architecture-support** root node method.

Architecture and Implementation Notes:

- Most of the **IBM,RPA-Client-Config** ELF header functionality is subsumed by the **ibm,client-architecture-support** root method. However, the **ibm,client-architecture-support** root method does not support the functionality specified through the ns.min-load field of the **IBM,RPA-Client-Config** ELF header. Supporting firmware implementations are prepared to move themselves out of the way when loading client programs.
- When booting a client program, firmware processes an **IBM,RPA-Client-Config** ELF header if present; a subsequent call of the **ibm,client-architecture-support** root method with conflicting values in the **ibm,architecture.vec** structure, overrides the configuration variables set by the ELF header.

Formal definition of ibm,architecture.vec:

ibm,architecture.vec = a **PVR-list** : **Number-of-option-vectors** : **option-vectors**[Number-of-option-vectors + 1]

PVR-list = **Terminator-list-entry** | **Non-terminator-list: Terminator-list-entry**

Non-terminator-list = Non-terminal-list-entry | **Non-terminal-list-entry : Non-terminator-list**

List-entry = 4-byte-mask : 4-byte-PVR-value

Terminator-list-entry = **List-entry** such that ! **4-byte-mask** & **4-byte-PVR-value != 0x00000000**

Non-terminator-list-entry = List-entry such that ! **4-byte-mask & 4-byte-PVR-value == 0x00000000**

Number-of-option-vectors = The number of option vectors is n+1 where n is the numeric value of the byte (byte value of 0x00 represents one option vector)

option-vector (option-vectors number 1-255): 1 byte length of the option vector where the number of bytes in the option vector (including the first byte of length) is n+2 where n is the numeric value of the byte (byte value of 0x00 represents a two byte option vector -- one length byte and one bit-vector byte) followed by 1-256 bytes of **bit-vector**.

option-vector (option-vector number 256): is special in that it is reserved for expansion. The first byte is again the number of option vectors in the vector expansion (see definition of **Number-of-option-vectors** above). This is followed by 1-255 **option-vectors** (see definition above) and potentially a 256th **option-vector** which is again an expansion option vector, and so on.

bit-vector: The structure of a bit vector is vector specific, in general support for most options are indicated by setting a specific bit to a 1, see Table 244, “ibm,architecture.vec option vectors,” on page 681.

The **PVR-list** of the **ibm,architecture.vec** structure is processed for the PVR value of each processor that the client program may be exposed to until either a **List-entry** allows the process to continue, or the **Terminator-list-entry** has been processed. If no **List-entry** allows the process to continue, then the **ibm,client-architecture-support** method terminates partition operation as with a boot failure. A **List-entry** allows the process to continue if either of the two following conditions hold.

1. (Processor-PVR-value & **List-entry[4-byte-mask]) == (List-entry[4-byte-PVR-value] & List-entry[4-byte-mask])** /*The client program explicitly supports the processor implementation */
2. If (the processor requires no client support for errata) && **(Logical-Processor-PVR-value & List-entry[4-byte-mask]) == (List-entry[4-byte-PVR-value] & List-entry[4-byte-mask])** /* Client program specifies support for this level of architecturally compliant processors */

`List-entry` values of special interest (these are **`Terminator-list-entry`** values):

- 0x00000000 0xFFFFFFFF Single entry list that matches any PVR value
- 0xFF000000 0x0FFFFFFF Single entry list that matches all architecturally compliant processors.

Table 244. ibm,architecture.vec option vectors

Option Array	Option Vector	Byte Number	Bit Number	Description
Base	1 PowerPC Server Processor Architecture Level [6]	1	0	Ignore [1]
			1	Cessation Policy [2]
			2	Reserved for Expansion (0b0)
			3	
			4	
			5	
			6	
			7	
		2	0	2.00
			1	2.01
			2	2.02
			3	2.03
			4	2.04
			5	2.05
			6	2.06
			7	2.07
		3	0	2.08
			1-7	Reserved for Expansion (0b0)
		4-256	Reserved for Expansion	

Table 244. ibm,architecture.vec option vectors ***(Continued)***

Option Array	Option Vector	Byte Number	Bit Number	Description
Base	2 Open Firmware	1	0	Ignore [1]
			1	Reserved
			2	real-mode
			3	Reserved for Expansion (0b0)
			4	
			5	
			6	
			7	
		2-3	0-15	Reserved for Expansion (0x0000)
		4-7 real-base	0-31	OF real starting address or -1 for platform default
		8-11 real-size	0-31	Maximum OF size or -1 for platform default
		12-15 virt-base	0-31	OF starting virtual address or -1 for platform default (valid for real-mode = 0)
		16-19 virt-size	0-31	Maximum OF virtual size or -1 for platform default (valid for real-mode = 0)
		20-23 load-base	0-31	Starting address of the client program load or -1 for platform default
		24-27 min-rma-size	0-31	Minimum size of RMA in MB[3] (total bytes = N*(2**20))
		28-31 min-load	0-31	Minimum client code to load at load-base or -1 for full client program at load base
		32 min-rma%	0-8	RMA size => M% * Partition_memory_size where M is the value of this 8 bit field[3]
		33 max-pft-size	0-8	The maximum size of the hash page table as 2**n 17<n<46
		34-256	Reserved for Expansion	

Table 244. ibm,architecture.vec option vectors ***(Continued)***

Option Array	Option Vector	Byte Number	Bit Number	Description
Base	3 IBM PowerPC Server Processor Options[6]	1	0	Ignore [1]
			1	Cessation Policy [2]
			2	Reserved for Expansion (0b0)
			3	
			4	
			5	
			6	
			7	
		2	0	Floating Point
			1	VMX
			2	Decimal Floating Point
			3	Decimal Floating Point Facility (The value of the ibm,dfp property indicates the architecture level of the facility.)
			4	Reserved for Expansion (0b0)
			5	
			6	
			7	
		3-256		Reserved for Expansion
Base	4 LoPAPR Implementation	1	0	
			1	Cessation Policy [2]
			2	Reserved for Expansion (0b0)
			3	
			4	
			5	
			6	
			7	
		2	0-7	Minimum VP entitled capacity percentage * 100 (if absent assume 10%)
		2-256		Reserved for Expansion

Table 244. ibm,architecture.vec option vectors *(Continued)*

Option Array	Option Vector	Byte Number	Bit Number	Description
Base	5 LoPAPR or OF Options [5]	1	0	Ignore [1]
			1	Cessation Policy [2]
			2	Reserved for Expansion (0b0)
			3	
			4	
			5	
			6	
			7	
		2	0	Logical Partitioning: If set the client program supports logical partitioning and associated hcall()s; else the client program shall be run with the hypervisor bit on.[7]
			1	Shared Processor Logical Partitioning: If set the client program supports the Shared Processor LPAR Option and may be run with that option enabled; else the Shared Processor LPAR Option shall be disabled for this partition.
			2	ibm,dynamic-reconfiguration-memory: If set the client program supports the **"ibm,dynamic-reconfiguration-memory"** property and it may be presented in the device tree; else, the partition memory shall be represented with individual **memory** nodes.
			3	Large Pages: If this bit is set, the client supports pages larger than 4 KB; else, the platform shall represent all of memory as mapped via 4 K pages.
			4	Alpha Partition [4]
			5	Tolerate long delays in H_MIGRATE_DMA
			6	Client supports donating dedicated processor cycles
			7	PCI Express/MSI Support: If set, the client supports PCI Express implementations utilizing Message Signaled Interrupts (MSIs).
		3	0	On input to ibm,client-architecture-support a non-zero value indicates that the client supports the I/O Super Page Option (Support of >4K I/O pages) (Includes extensions to H_MIGRATE_DMA for >4K I/O pages and >256 xlates). See Section 14.5.4.8, "Memory Migration Support hcall()s," on page 431. In the **ibm,architecture-vec-5** property of the **/chosen** node, a non-zero value indicates that the platform supports the I/O Super Page Option (Support of >4K I/O pages).
			1-4	On input to ibm,client-architecture-support this field shall be zero. In the **ibm,architecture-vec-5** property of the **/chosen** node, this field represents the implementation dependent number of xlates entries supported per migration operation as: 256 * 2**N. See Section 14.5.4.8, "Memory Migration Support hcall()s," on page 431.
			5-7	On input to ibm,client-architecture-support this field shall be zero. In the **ibm,architecture-vec-5** property of the **/chosen** node, this field represents the implementation dependent number of simultaneous migration options supported as: 2**N. See Section 14.5.4.8, "Memory Migration Support hcall()s," on page 431.

Table 244. ibm,architecture.vec option vectors ***(Continued)***

Option Array	Option Vector	Byte Number	Bit Number	Description
Base	5 LoPAPR or OF Options [5]	4		Cooperative Memory Over-commitment Option Control
			0	The value of 1 enables the Cooperative Memory Over-commitment Option
			1	The value of 1 enables the Extended Cooperative Memory Over-commit Option
			2-7	Reserved for Expansion
		5		Associativity Information Option Control
			0	= the "Form value" of the **"ibm,associativity"** and **"ibm,associativity-reference-points"** properties. See Chapter 15, "Non Uniform Memory Access (NUMA) Option," on page 505 for further details.
			1	Platform Resource Reassignment Notification (Affinity Change)
			2-7	Reserved for Expansion
		6		Binary Option Controls
			0	Enable MTT Option See Section 14.5.4.2.4, "H_PUT_TCE_INDIRECT," on page 421.
			1	Reserved
			2	Enable Active Memory Compression
			3	Enable Universlly Unique IDentifier Option (UUID)
			4	Reserved for Expansion
			5-7	Reserved for Expansion
		7		Reserved for Expansion
		8		Reserved for Expansion
		9-12		Max Processors Supported (For legacy support, if this byte is not present the partition is limited to a maximum of 64 processors)
			0-31	32 bit unsigned integer maximum number of OF device tree nodes of type "cpu" that may be presented to this partition.
		13-14	0-7 & 0-7	Highest Base LoPAPR Level Supported as the binary contents of 13.14 (i.e. level 4.15 would be encoded as 0x040F)
		15-16		Reserved for Expansion
Base	5 LoPAPR or OF Options [5]	17-20		Platform Facilities Enable – Value of 0b1 indicates facility is enabled
			0	Random Number Generator
			1	Compression Engine
			2	Encryption Engine
			3-31	Reserved for Expansion -- Value = 0b0

Table 244. ibm,architecture.vec option vectors ***(Continued)***

Option Array	Option Vector	Byte Number	Bit Number	Description
		21	0-7	Sub-Processor Representation Level -- Defined Values: 0: Sub-Processors not supported 1: 1,2,or 4 Sub-Processors supported 2-255 Reserved
		22-256		Reserved for Expansion
Base	6 Hints	1	0	Reserved for Expansion (0b0)
			1	
			2	
			3	
			4	
			5	
			6	
			7	
		2	0	Secondary Page Table Entry Group: If set, the client does not use secondary page table entry groups; else the client may use secondary page table entry groups.
			1	Reserved
			2	
			3	
			4	
			5	
			6	
			7	
		3	0-7	OS Name: Represents the name of the client OS. Defined values include: 0x0: Reserved 0x1: AIX 0x2: Linux 0x3-0xFF: Reserved for Expansion
		4-256	Reserved for Expansion	
	7 OS Identification	1-256	An ASCII character formatted null terminated string that describes the client operating system. The string shall be human readable and may be displayed on the console.	
	8-255	Reserved for Expansion		
	256	Reserved for Expansion to the first Extension Option Array		
Extensions 1-N	Reserved for Expansion			

Notes:

1. The Ignore Policy bit indicates that the client program assumes all responsibility for the options represented by the option vector. The firmware is to configure the platform at the highest level consistent with its configuration variables and ignore the rest of the specific option vector. An option vector with the Ignore Policy bit set need be no longer than two bytes (size=0, data = 0b1ddd dddd where d = don't care).

2. The Cessation Policy Bit determines if the partition continues to run if the platform must operate with an option enabled that is not explicitly supported by the client program as represented by the option vector setting. If the Cessation Policy Bit is 1, then processing halts as with a boot failure. If the Cessation Policy Bit is 0 then client program processing continues if the unsupported option is initialized to a benign state and stays in that state unless an aware program activates the option, and the option does not appear in the device tree. If an unsupported option cannot be initialized to a benign state, then processing halts with a boot failure. Following are the detailed definitions of benign state for selected bit vectors.

   - For option vector numbers 1 "PowerPC Server Processor Architecture Level" and 3 "IBM PowerPC Server Processor Extensions" the benign state is defined as unable to generate exceptions, mask errors, or present covert channel exposures.
   - For option vector number 5 "LoPAPR Options" Byte 2 bit 5 "Alpha Partition" The Cessation Policy bit is not applicable.
   - For option vector number 2 "Open Firmware" the Cessation Policy Bit is not defined, the platform either accommodates the values defined in the option vector or proceeds as with boot failure.

3. The Initial size of the RMA is set to the greater of the values indicated by bytes 24-27 or 32 of option vector number 2 "Open Firmware" or minimum RMA size supported by the platform and capped by the maximum memory defined for the partition and the maximum size of the RMA supported by the platform. The respective selected values are reported in the length of the first **`memory`** property.

4. The Alpha flag only applies to the first partition of a non HMC managed system and activates overrides to the partition's I/O resource allocation as defined in the partition definition.

   - If the system is HMC managed, the flag is ignored and the client program gets the resources assigned by its partition definition (no overrides are activated).
   - If the partition is not the first partition, the flag is ignored and the client program gets the resources assigned by the partition definition (no overrides are activated).
   - If the Alpha flag applies, and is set, then the partition gets a VMC virtual I/O device in its device tree regardless of its partition definition (Override to include VMC is activated).
   - If the Alpha flag applies, and is not set, then the partition does not get a VMC virtual I/O device in its device tree regardless of its partition definition (Override to remove VMC is activated) and it gets all the physical I/O resources in its device tree regardless of its partition definition (Override to include all physical I/O is activated). Note this condition requires that any other platform partitions be terminated.

5. Given that the Ignore policy bit is off and the partition continues to run, the options and values presented in by this option vector and supported/chosen by the platform firmware are reported in the **`"ibm,architecture-vec-5"`** property of the **`/chosen`** node.

6. Option vector number 1 "PowerPC Server Processor Architecture Level" and the property that reports the chosen value (i.e., **`"cpu-version"`**) represent the operational base architectural level of the processors -- that is without regard to enabled processor architectural options. Option vector number 3 "IBM PowerPC Server Processor Extensions" and option specific properties that report the chosen values represent the active processor architectural options. Some processor implementations may not support all combinations of these two option vectors. The firmware shall activate the highest level of processor support, consistent with partition attributes, that does not exceed the most restrictive of the two option vectors. Note the Cessation Policy bit may allow operation where the lowest level of processor support still exceeds the most restrictive case.

7. If a client program does not support logical partitioning no other client programs may be running simultaneously on the platform. The platform may impose further restrictions beyond the scope of LoPAPR. If the platform honors the client program restriction of not supporting logical partitioning, upon return the logical real address equals the platform real address. If the platform can not honor the restriction, the processing terminates as with a boot failure. The cessation policy option vector bit has no effect upon logical partitioning option vector bit.

B.6.2.4 ROM Node(s)

The ROM Node(s), when present to represent optional platform read only memory containing directly executable platform firmware, shall be a child or children of the root node.

B.6.2.4.1 ROM Node Properties

Each ROM Node shall have the following properties:

"name" S

Standard *property name* that denotes a ROM Node.

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

The value of this property shall be "rom".

"reg" S

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

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

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

"#address-cells" S

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

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

"ranges" S

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

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

"available" S

Standard *property name* to define available ROM resources.

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

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

"write-characteristic" S

Standard *property name* defines the ROM Technology.

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

"cacheable" S

OF standard property indicating that the ROM is cacheable.

prop-encoded-array: <none>.

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

B.6.2.4.2 ROM Node Methods

If one or more ROM nodes are present, they shall each implement the following standard methods per *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], Section 3.6.1. The **"reg"** property is used to determine which ROM the standard methods apply to for multiple ROM's.

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

open (-- true) M

Standard method to prepare the ROM Node for subsequent use.

close (--) M

Standard method to close the previously opened ROM Node.

decode-unit (addr len -- phys.lo...phys.hi M

Standard method to convert text unit-string to physical address.

encode-unit (phys.lo...phys.hi -- unit-str unit-len) M

Standard method to convert physical address to text unit-string.

probe (--) M

OF method used at boot time to probe all ROM's.

The **probe** method for ROM Nodes shall probe for FCode images within the address space defined by its **"reg"** property as defined herein. For each page within its address space, look for a valid FCode image. A valid FCode image is defined to start with an FCode-header (see section 5.2.2.5 in *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2]) where the first byte is **start1**, the format byte is 0x08, the length field indicates that the FCode program is contained within the address space of the **/rom** node, and where the checksum is correct. (This probing must take into account the possibility that the ROM image is in the opposite endian-ness from which OF is currently running.)

If such an FCode image is found, a new child node shall be created by executing **new-device** and **set-args**, the FCode image copied to memory (taking into account the endian-ness) and the copy evaluated with **byte-load**. (The FCode program can use **my-unit** to create its **"reg"** property.). The arguments used by **set-args** are defined to be 0,0,unit-str, unit-len where unit-str is a text string representation of the physical address location for the FCode Image and unit-len is the length of the FCode Image.

B.6.2.5 ROM Child Node(s)

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

B.6.2.5.1 ROM Child Node Properties

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

"name" S

Standard *property name* that denotes a ROM child node.

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

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

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

"reg" S

Standard *property name* that defines the child node address range for a ROM image(s).

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

Phys-addr is encoded as with **encode-phys**, and *size* is encoded as with **encode-int**. The *phys-addr* is a base address of the ROM image and *size* is the length of the ROM image.

B.6.2.5.2 ROM Child Node Methods

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

open (-- true) M

Standard method to prepare this device for subsequent use.

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

close (--) M

Standard method to close the previously opened device.

load (addr -- len) M

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

B.6.3 Run Time Abstraction Services (RTAS) Node

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

B.6.3.1 RTAS Node Properties

This section describes the *rtas* node properties.

"name" S

Standard *property name* that denotes the RTAS node.

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

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

`"rtas-event-scan-rate"` S

property name that is the rate at which an OS should read indicator/sensor/error data

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

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

`"rtas-indicators"` S

property name that indicates indicators are implemented.

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

The values for this property is a list of integers that are the token values (*token*) for the defined indicators and the number of indicators (*maxindex*) for that token which are implemented (see Chapter 7, "Run-Time Abstraction Services," on page 107) on the platform.

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

`"rtas-sensors"` S

property name that indicates sensors are implemented.

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

The values for this property is a list of integers that are the token values (*token*) for the defined sensors and the number of sensors (*maxindex*) for that token which are implemented (see Chapter 7, "Run-Time Abstraction Services," on page 107) on the platform.

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

`"rtas-version"` S

property name describes version information for the RTAS implementation.

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

The value of this property shall denote the version the RTAS implementation. For this version, the integer shall be as defined in this architecture.

`"rtas-size"` S

property name is the size of the RTAS memory image.

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

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

`"rtas-display-device"` S

property name identifies RTAS Display Device

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

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

"rtas-error-log-max" S

property name identifies maximum size of an extended error log entry.

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

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

"power-on-max-latency" S

property name specifies a future power on time capability.

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

The value of this property specifies the capability of the hardware to control the delay of system power on in days. If the property is present, the value shall indicate the maximum delay or latency in days. If the property is not present, the maximum delay or latency is 28 days.

"ibm,preserved-storage"

property name specifies that the client program was loaded with one or more LMBs preserved from a previous client program.

prop-encoded-array: None, this is a name only property.

The client program may wish to save the contents of the preserved LMBs and deregister the LMBs for preservation.

"ibm,scan-log-directory"

property name specifies that the platform supports the scan-log directory option.

prop-encoded-array: None, this is a name only property.

"ibm,indicator-<*token*>**"**

property name to provide a FRU location code for identifying each indicator.

prop-encoded-array: an array of *maxindex* + 1 strings, encoded as with **encode-string**.

"ibm,sensor-<*token*>**"**

property name to provide a FRU location code for identifying each physical sensor.

prop-encoded-array: an array of *maxindex* + 1 strings, encoded as with **encode-string**.

"ibm,display-line-length"

property name to provide the length of a display line in number of characters.

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

"ibm,display-number-of-lines"

property name to provide the number of lines in the display.

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

"ibm,display-truncation-length"

property name, when provided, specifies the length to which each line to be display is truncated, based on which line of the physical display on which the line is displayed. When the truncation length is greater than the length

specified in the **"ibm,display-line-length"** property, then the platform provides a platform-dependent method of displaying the line to the user.

prop-encoded-array: An array of integers, each encoded as with encode-int. The number of integers corresponds to the number of lines, as defined by the **"ibm,display-number-lines"** property. The first integer refers to the truncation length for the first physical line of the display, the second to the second physical line, and so on.

"ibm,form-feed"

property name to provide an indication of the form-feed capability.

prop-encoded-array: a character, NULL (0x00) if form-feed is not supported and np (0x0C) if form-feed is supported, encoded as with **encode-int**.

"ibm,environmental-sensors"

property name describes the environmental sensors which are available to an application.

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

"ibm,flash-block-version"

property name in the **/rtas** node indicates the block list format to be used.

prop-encoded-array: integer encoded as with **encode-int**. Value is 0x01 for the discontiguous block list. (If a new version of the block list is ever required, other values could be defined.)

"ibm,errinjct-tokens" S

property name, defines the error inject functions implemented on this platform.

prop-encoded-array: List of *(errinjct-token-name, errinjct-token-value)* specifications.

errinjct-token-name: A string, encoded as with **encode-string**.

errinjct-token-value: is encoded as with **encode-int**.

"ibm,lrdr-capacity"

property name in the **/rtas** node identifies the dynamic reconfiguration capabilities of the partition

prop-encoded-array: A triple consisting of phys, size, and one integer encoded as with **encode-int**

The phys (of size #address-cells) communicates the maximum address in bytes and therefore, the most memory that can be allocated to this partition.

The size (of size #size-cells) communicates the increment (quantum of logical memory dynamic reconfiguration).

The first integer communicates the maximum number of processors (implied quantum of 1).

Note: Some implementations depend upon the presence and value of a second integer. Future extensions to this property should not define a second integer for new purposes.

"ibm,hypertas-functions"

property name, of the **/rtas** node, defines the platform's implemented hypervisor RTAS function sets.

prop-encoded-array: List of *Hypervisor-RTAS-function-set* specifications.

Each *Hypervisor-RTAS-function-set* specification is a byte string encoded as with **encode-string**.

"ibm,dma-delay-time"

property name to define the time delay need to ensure outstanding DMA operations targeting migrated pages have completed.

prop-encoded-array: A one cell integer encoded as with **encode-int** that represents the number of micro-seconds that the OS should wait prior to reusing migrated DMA read target pages.

"ibm,associativity-reference-points" S

property name to define the associativity reference points for the **"ibm,associativity"** properties of this platform.

prop-encoded-array: A list of one or more integers cell(s) encoded as with **encode-int**.

"ibm,max-associativity-domains"

property name to define the maximum number associativity domains for this platform.

prop-encoded-array: An associativity list such that all values are the maximum that the platform supports in that location. The associativity list consisting of a number of entries integer (N) encoded as with **encode-int** followed by N integers encoded as with **encode-int** each representing maximum number associativity domains the platform supports at that level.

"ibm,request-partition-shutdown"

property name to specify that the partition was rebooted in the forced fire hose dump mode.

prop-encoded-array: An integer encoded as with **encode-int** that represents the platform's partition shutdown configuration variable. The defined states are:

0 = The platform boots with no request to save appropriate data nor shutdown the partition.

1 = The platform boots with a conditional request to save appropriate data and shutdown the partition. The client program should check for an EPOW sensor state of 3 and if present, it should save appropriate data and shutdown the partition. If the EPOW sensor state of 3 is not present, then the partition should initiate a reboot since the device tree will be incomplete.

2 = The platform boots with a mandatory request to the client program to save appropriate data and shutdown the partition.

"ibm,integrated-stop-self"

property name indicating that prior to placing a processor in the stopped state, the platform flushes and disables any caches/memory exclusively used by the issuing processor.

prop-encoded-array: NULL

"ibm,rks-hcalls"

property name: indicating the hcalls that are implemented with a reduced kill set. Absence of this property indicates that only hcalls that are specified as always having a reduced kill set provide that semantic.

prop-encoded-array: A one to N byte bit vector, bit positions representing hcall()s (see Table 245, ""ibm,rks-hcalls" bit vector to hcall map.," on page 695) that present a reduced kill set per their architectural specification.

Table 245. **"ibm,rks-hcalls"** bit vector to hcall map.

Byte Number	Bit Number	hcall
0	0	0b11 for H_CONFER & H_PROD
	1	
	2	Set to 1 if H_PURR is implemented with a reduced volatile kill set of r3 & r4; else set to 0.
	3	Reserved for future expansion (0b0)
	4	
	5	
	6	
	7	
1-N	Reserved for future expansion	

"ibm,reset-capabilities"

property name indicates what capabilities the platform supports relative to the ibm,set-slot-reset RTAS call, when that RTAS call is implemented.

prop-encoded-array: An integer encoded as with encode-int that represents the functions supported in the *ibm,set-slot-reset* RTAS call

0 = Platform supports Functions 0 and 1 supported.

1 = Platform supports Functions 0, 1, and 3.

Note: The absence of this property implies the platform supports Functions 0 and 1 for the *ibm,set-slot-reset* RTAS call, when that RTAS call is implemented.

"ibm,configure-kernel-dump-sizes"

property name specifies that the Platform Assisted Kernel Dump option is supported for sections described by this property.

prop-encoded-array: For each dump section type supported, a 32 bit cell which defines the ID of a supported section followed by two 32-bit cells which gives the size of the section in bytes (not including any disk headers.)

"ibm,configure-kernel-dump-version"

property name specifies that the Platform Assisted Kernel Dump option is supported for versions described by this property.

prop-encoded-array: Contains a 16-bit cell describing the minimum kernel dump version supported by the firmware followed by a 16-bit cell describing the maximum kernel dump version supported by the firmware.

"ibm,kernel-dump"

property name specifies the presence of a registered kernel dump in the Platform Assisted Kernel Dump option.

prop-encoded-array: Contains the description of the registered kernel dump in the format described in Table 124, “Kernel Assisted Dump Memory Structure,” on page 256.

“ibm,read-slot-reset-state-functions”

property name specifies the implementation of certain input or output fields in the *ibm,read-slot-reset-state2* RTAS call. If this property does not exist, then the *ibm,read-slot-reset-state2* RTAS call implements only the first 3 inputs and the first 4 outputs (*Number Inputs* is required to be 3 and the *Number Outputs* is required to be 4), as defined in Table 77, “ibm,read-slot-reset-state2 Argument Call Buffer,” on page 185.

prop-encoded-array: Contains a 32 bit cell, with the bits defined as follows:

Bits 0-29: Reserved (value of 0).

Bit 30: When a value of 1, the *ibm,read-slot-reset-state2* RTAS call checks the *Number Outputs* and the implements the 5th output (*Number Outputs* of 5), as defined by Table 77, “ibm,read-slot-reset-state2 Argument Call Buffer,” on page 185.

Bit 31: When a value of 1, the *ibm,read-slot-reset-state2* RTAS call implements the first 3 inputs and the first 4 outputs (*Number Inputs* of 3 and the *Number Outputs* of 4), as defined in Table 77, “ibm,read-slot-reset-state2 Argument Call Buffer,” on page 185. This bit is always required to be a value of 1 when this property is implemented.

“ibm,change-msix-capable”

property name indicating the platform supports the *ibm,change-msi* RTAS call with *Number of Outputs* equal to 4 and *Functions* 3 and 4.

prop-encoded-array: <none>

B.6.3.2 /RTAS node DR Sensors and Indicators

The following sensors and indicators are defined for the /RTAS node for the DR option.

“9003”

sensor token, the existence of this token number denotes that the platform supports the 9003 “DR entity sense” sensor.

“9001”

indicator token, the existence of this token number denotes that the platform supports the 9001 “isolation state” indicator.

“9002”

indicator token, the existence of this token number denotes that the platform supports the 9002 “dr-indicator” indicator used to guide operators in the physical add or removal of hardware.

“9003”

indicator token, the existence of this token number denotes that the platform supports the 9003 “allocation-state” indicator.

“ibm,extended-os-term”

property-name indicating that the platform supports extended *ibm,os-term* behavior as described in Section 7.3.9.1, “ibm,os-term,” on page 165.

prop-encoded-array: encode-null

B.6.3.3 RTAS Function Property Names

This section defines the property names associated with the various RTAS functions defined by Table 17, “RTAS Tokens for Functions,” on page 112. Table 17 should be used as the reference for RTAS Functions currently implemented. Each RTAS function that a platform implements shall be represented by its own function property, who’s value is the *token* used to invoke the function on an RTAS call.

The formal property definition for each such property is of the form:

property name specifies the name of the RTAS function -- such as:

“nvram-fetch” S

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

If an RTAS function is implemented, there is a property name which corresponds to its function name. The value of this property is a *token*. This *token*, when passed to RTAS via its *rtas-call* interface (see below), invokes the named RTAS function. If a RTAS function is not implemented, there will not be a property corresponding to that function name. See the Chapter 7, “Run-Time Abstraction Services,” on page 107 for more information about RTAS functions.

“ibm,termno”

property name of the **/rtas** node defines the virtual terminal numbers available for use by this partition.

prop-encoded-array: A pair of integers encoded as with **encode-int**, the first being the value of the lowest termno in a contiguous range of supported values, the second being the number of termno values supported.

Note: The number of supported termno values is implementation dependent -- the minimum number is one.

“ibm,hypertas_functions”

property name of the /RTAS node, defines the platform’s implemented hypervisor RTAS function sets.

prop-encoded-array: List of *Hypervisor-RTAS-function-set* specifications.

Each *Hypervisor-RTAS-function-set* specification is a byte string encoded as with **encode-string**.

B.6.3.4 RTAS Node Methods

The **instantiate-rtas** or **instantiate-rtas-64** method is invoked by the OS to instantiate the RTAS functionality. This is accomplished via the call-method *Client Interface Service*. If the platform supports the **ibm,client-architecture-support** root node method, and that method has not been called prior to the call of the **instantiate-rtas** or **instantiate-rtas-64** methods, then the platform shall operate at the least functional level supported by the platform.

Note: Platforms should provide a manual override capability to allow most functional level allowed by the partition configuration in the event that a client program does not call the **ibm,client-architecture-support** root node method prior to the instantiation of RTAS.

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

Invoking the **instantiate-rtas** method binds the RTAS environment to a given location in System Memory and initializes the RTAS environment. The in parameter, *rtas-base-address*, is the physical address to which the RTAS environment is to be bound. This call indicates that RTAS is instantiated in a 32-bit mode. The amount of

contiguous real memory that should be allocated for the RTAS environment is given by the value of the **"rtas-size"** property.

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

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

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

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

B.6.4 Properties of the Node of type cpu

When the platform implements the LPAR option the following properties are required of the /cpus node

ibm,pft-size

property name of the children of type "cpu" of the **/cpus** node, defines the size of the processor's page frame table.

prop-encoded-array: A pair of integers encoded as with **encode-int**, the first being the NUMA CEC Cookie (up to a maximum of (2^{16})-1) the second being the base 2 log of the size of the page frame table in bytes.

Notes:

1. On single CEC platforms, the NUMA CEC Cookie value is zero.
2. Due to constraints caused by initial memory allocations, and other running partitions, the firmware may not be able to allocate a node's PFT for the full size requested in the PFT_size configuration variable. The **"ibm,pft-size"** property of course reflects the actual size allocated.
3. The partitions running on multiple NUMA nodes would have multiple PFTs which did not look alike due to the difference in mapping local and remote page frames.)

To support dynamic addition and removal of processors, the /cpus node contains the properties: *ibm,drc-types* (cpu), *ibm,drc-indexes ibm,drc-names* and *ibm,drc-power-domains* (-1's). These properties have entries for the maximum number of dynamically reconfigurable processors that the platform supports for the specific OS image.

"ibm,ppc-interrupt-server#s" S

property name: Defines the single processor server numbers associated with this processor. Placing the numerical equivalent of one of these quantities into the server# field of an XIVR directs associated interrupts to this processor. The first server number is associated with the "primary processor thread" any subsequent numbers are associated with the secondary. etc. hardware threads that the processor may implement.

prop-encoded-array: A list of one or more integers in the range of 0 to **2"ibm,interrupt-server#-size"** encoded as with **encode-int**.

Note: In order to achieve optimal performance, processor server numbers should be activated in the order that they are presented in the **"ibm,ppc-interrupt-server#s"** property and deactivated in the reverse order.

"ibm,ppc-interrupt-gserver#s" S

property name: Defines the multiple processor global server numbers to which this processor belongs. Placing the numerical equivalent of one of these quantities into the server# field of an XIVR directs associated interrupts to one of the processors in that group.

prop-encoded-array: A list of (*server#, gserver#s*) specification pairs. the first integer specifies a single processor *server#* as presented in the node's **"ibm,ppc-interrupt-server#s"** property, followed by an integer with a value less than or equal to 2**"ibm,interrupt-server#-size"** encoded as with **encode-int** that specifies the global server queue that also may present interrupts to the interrupt management area associated with the *server#*.

"ibm,sub-processors"

property name: the sub-processor configuration that is running on this processor. In the absence of this property, this processor may not be divided into sub-processors.

prop-encoded-array: a series of three or more integers each encoded as with encode-int. The value of the first integer indicates how many integers follow (the value 2 indicates that two integers follow). The second integer indicates the number of sub-processors that are running on this processor. If the processor is not divided into sub-processors the value of the second integer shall be 1, two sub-processors shall be represented by the value 2, four sub-processors shall be represented by the value 4 and so on. The third integer indicates the maximum number of sub-processors that could be configured to run on this processor.

Client programs shall ignore subsequent integers beyond those defined at the time they were written.

B.6.5 Extensions for LoPAPR I/O Sub-Systems

LoPAPR I/O sub-system events may be signaled in a variety of ways depending upon platform capabilities. In order of increasing functionality:

1. Events are universally fatal, and are signaled via checkstop.
2. After being enabled, the effected section enters freeze state signalling this state with a return of all 1's to any MMIO load instruction (If not enabled functionality of the specific section reverts to #1. Presence of *ibm,set-eeh-option* RTAS call denotes platforms that support this level of functionality.)
3. An extension to #2 above wherein, after being enabled for a specific section of the I/O sub-system, additional event conditions may be reported and events are signaled using an external interrupt. The platform's capability to support this level of functionality is reported by the inclusion of the **"ibm,i/o-events-capable"** property (see definition below) in nodes where enabling control may be exercised.

"ibm,i/o-events-capable"

property name indicating that I/O sub-system events detected by the hardware represented by this node in the device tree may be singled with an I/O event interrupt if enabled.

prop-encoded-array: 0 to N interrupt specifiers (per the definition of interrupt specifiers for the node's interrupt parent).

When no interrupt specifiers are present, then the interrupt, if enabled, is signaled via the interrupt specifier given in the **I/O-events** child node of the **/events** node.

To perform certain management functions, it is necessary to quiesce segments of the platform's I/O infrastructure, such quiescence domains are not representable by a strict tree structure. The **"ibm,io-quiesce-domains"** property relates the membership of the various elements of a platform's I/O sub-system to such quiescence domains.

"ibm,io-quiesce-domains"

property name indicating the I/O quiesce domains of which this device, and all devices under this device (if any), is a member.

prop-encoded-array: List of one or more domain-id's to which this device belongs, and to which all devices under this device (if any) belongs. Domain-id's are encoded as with **encode-int**.

Virtual I/O that does not take up physical address locations is represented in a device sub tree for which the **"#size-cells"** and **"#address-cells"** properties are zero and one, respectively. However, the ibm dma-window properties, such as **"ibm,dma-window"** and **"ibm,my-dma-window"**, need to contain real size and address fields. The number of cells for these real size and address fields need to be non-zero.

"ibm,#dma-size-cells"

property name to define the package's dma address *size* format.

prop-encoded-array: number encoded as with **encode-int**.

The property value specifies the number of cells that are used to encode the size field of ibm dma-window properties. If the **"ibm,#dma-size-cells"** property is missing, the default value is the **"#size-cells"** property for the package. If both the **"ibm,#dma-size-cells"** and **"#size-cells"** properties are missing, refer to the **"#size-cells"** property definition in the *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2] for the default value.

"ibm,#dma-address-cells"

property name to define the package's dma address format.

prop-encoded-array: number encoded as with **encode-int**.

The property value specifies the number of cells that are used to encode the physical address field of ibm dma-window properties. If the **"ibm,#dma-address-cells"** property is missing, the default value is the **"#address-cells"** property for the package. If both the **"ibm,#dma-address-cells"** and **"#address-cells"** properties are missing, refer to the **"#address-cells"** property definition in the *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2] for the default value.

B.6.5.1 PCI Host Bridge Nodes

This section describes the PCI Host Bridge (PHB) properties which are added or modified for an LoPAPR implementation. Refer to *PCI Bus binding to: IEEE Std 1275-1994, Standard for Boot (Initialization, Configuration) Firmware* [6] for the base PCI properties and methods. For each platform PCI Host Bridge, a **"reg"** property shall be present in the respective PCI Node.

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

To support dynamic addition and removal of PHBs, the / node contains the properties: *ibm,drc-types* (phb), *ibm,drc-indexes ibm,drc-names* and *ibm,drc-power-domains* (-1's). These properties have entries for the maximum number of dynamically reconfigurable PHBs that the platform supports for the specific OS image.

B.6.5.1.1 PCI Host Bridge Properties

For each PHB in the platform (called a PCI Bus Controller in the *PCI Bus binding*), a PCI Host Bridge Node shall be defined as a child node of the system bus, in accordance with *PCI Bus binding to: IEEE Std 1275-1994, Standard for Boot (Initialization, Configuration) Firmware* [6]. Each PCI PHB Node shall have a Unit Address defined in the **"reg"** property that is unique and persistent from each boot-to-boot. One way for the platform to meet this requirement is to supply a virtual Unit Address based upon a unique identifier stored in the hardware. In this case, the size field of the first **"reg"** property phys-address, size pair shall be zero. The following properties are modified or added by this architecture and shall apply to each of these nodes.

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

"ranges" S

Standard *property name* defines this PHB's physical address ranges.

prop-encoded-array: Two or more (*child-phys, parent-phys, size*) specifications.

This property is mandatory for PCI Host Bridges in LoPAPR implementations. The property value consists of four (*child-phys, parent-phys, size*) specifications, as described in *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2].

The first specification shall specify the configured address and size of this PHB's I/O Space. (I/O Space is shown as "BIOn" to "TIOn" in Chapter 3, "Address Map," on page 59.) The second specification shall specify the configured address and size of this PHB's Memory Space. (Memory Space is shown as "BPMn" to "TPMn" in the Common Hardware Reference Platform Architecture.)

"model" S

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

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

The value of this property is a vendor dependent string which uniquely identifies this PHB and is correlated to its manufacturer, part number, and revision level. (see *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2] for more information.) The string value is device dependent, but shall supply information sufficient to identify the part to a level equivalent to the level achievable via the standard PCI configuration registers: **"vendor-id"**, **"device-id"**, and **"revision-id"**.

"64-bit-addressing" S

property name indicates this PHB's capability to address more than 4 GB of memory.

prop-encoded-array: <none>

This property shall be present indicating that the PHB supports addressing more than 4 GB of memory (required for all **PHB** nodes).

"external-control" S

property name indicates this PHB's ability to support the PA external control facility.

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

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

"ibm,tce-alloc-info"

property name indicates the addresses of platform pre allocated TCE table space.

prop-encoded-array: One to N *phys-addr, size* pair(s). The first pair represents the memory area allocated by the platform for the TCE tables associated with this PHB. Any subsequent pairs represent memory areas that the OS should avoid using to minimize performance impacts.

Phys-addr is encoded as with **encode-phys** the number of cells for *phys* corresponds to **"#address-cells"** value applicable to this node.

size the number of cells for *size* corresponds to the **"#cell-size"** value applicable to this node.

"ibm,max-completion-latency"

property name indicates the maximum DMA Read completion latency for IOAs under this PHB.

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

This property, when present (for example, see Requirement R1–9.1.10–2), indicates the maximum DMA Read completion latency for IOAs under this PHB, in microseconds. For plug-in adapters, the latency value does not include latency of any additional PCI fabric (for example, PCI Express switches) on the plug-in adapter.

"ibm,extended-address" S

property name indicates this platform supports Peripheral Memory Spaces, Peripheral I/O Spaces, and SCA spaces above 4 GB.

prop-encoded-array: <none>

This property must be present in all PHB nodes.

"ibm,pcie-link-width-stats"

property name indicates the collection of PCI Express link-width capabilities and measurements at the PE below the PHB.

prop-encoded-array: 2 integers encoded with **encode-int**

The first integer represents the maximum PCI Express lane-width at the Partitionable Endpoint.

The second integer represents the actual PCI Express lane-width at the Partitionable Endpoint.

Implementation Note: In some cases, a PCIe device may train at a different width depending on the speed capabilities of the link.

"ibm,pcie-link-speed-stats"

property name indicates the collection of PCI Express link-speed capabilities and measurements at the PE below the PHB.

prop-encoded-array: 2 integers encoded with **encode-int**. The format of each integer is identical to the link speed encodings defined in the PCI Express Capability Structure chapter of the *PCI Express Base Specification* [22]. In the 2.0 version of that specification, it defines 0x1 = 2.5 GT/s and 0x2 = 5.0 GT/s.

The first integer represents the maximum PCI Express link-speed at the Partitionable Endpoint.

The second integer represents the actual PCI Express link-speed at the Partitionable Endpoint.

B.6.5.1.1.1 Properties for Children of PCI Host Bridges

The following properties are defined for PCI host bridges and their children.

"133mhz-capable" S

property name: The presence of this property indicates the device's capability of operating at 133 megahertz. Only present if PCI-X Status Register bit 17 is set.

prop-encoded-array: None.

"266mhz-capable" S

property name: The presence of this property indicates the device's capability of operating at 266 megahertz. Only present if PCI-X Status Register bit 30 is set.

prop-encoded-array: None.

"533mhz-capable" S

property name: The presence of this property indicates the device's capability of operating at 533 megahertz. Only present if PCI-X Status Register bit 31 is set.

prop-encoded-array: None.

"ibm,msi-ranges"

property name: Defines the Message Signaled Interrupt interrupt source number (as returned by H_XIRR) range(s) assigned to this unit using the MSI capability structure. (Note this property is only supplied if the package is assigned one or more message signaled interrupt numbers at boot time using the MSI capability structure, those packages assigned level sensitive interrupts include the standard interrupts property.) The platform firmware assigns the interrupt source numbers in order to the first N Message Signaled Interrupt configuration spaces of the adapter, setting the associated configuration spaces, in accordance with the platform's hardware configuration, to produce the interrupt source numbers specified.

prop-encoded-array: List of one or more (int-number, range) specifications.

Int-number is the first interrupt source number in a contiguous range of interrupt source numbers encoded as with **encode-int**.

Range is the one based count of consecutive interrupt source numbers that compose the specified range of interrupt source numbers, encoded as with **encode-int**.

"ibm,msi-x-ranges"

property name defines the Message Signaled Interrupt interrupt source number (as returned by H_XIRR) range(s) assigned to this IOA function using the MSI-X capability structure. (Note this property is only supplied if the package is assigned one or more message signaled interrupt numbers at boot time using MSI-X capability structure, those packages assigned level sensitive interrupts include the standard interrupts property.) The platform firmware assigns the interrupt source numbers in order to the first N MSI-X vectors of the IOA function, setting the associated configuration spaces and MSI-X vectors, in accordance with the platform's hardware configuration, to produce the interrupt source numbers specified.

prop-encoded-array: List of one or more (int-number, range) specifications.

Int-number is the first interrupt source number in a contiguous range of interrupt source numbers encoded as with **encode-int**.

Range is the one based count of consecutive interrupt source numbers that compose the specified range of interrupt source numbers, encoded as with encode-int.

"ibm,req#msi"

property name: Defines the number of Message Signaled Interrupts requested by the adapter as communicated in its MSI capability structure. This number may be greater than the number of Message Signaled Interrupts actually assigned by the firmware.

prop-encoded-array: number of requested interrupts encoded as with **encode-int**.

"ibm,req#msi-x"

property name: Defines the number of MSI-X Interrupts requested by the adapter as communicated in the Table Size field of the MSI-X Capability Structure for the adapter. This number may be greater than the number of MSI-X interrupts actually assigned by the firmware.

prop-encoded-array: number of requested MSI-X interrupts encoded as with **encode-int**.

"ibm,connector-type"

property name to identify the connectors associated with a built-in IOA that supports wrap test. This property must be provided if there is more than one connector for the same IOA on the platform.

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

"ibm,wrap-plug-pn"

property name to provide the part number(s) of the wrap plug(s) required for testing built-in IOAs with the default connector or the connectors specified in **"ibm,connector-type"**. If this property is provided in the same node *with* an **"ibm,connector-type"** property, there is a one-to-one correspondence between the strings in each property. If this property is provided *without* an **"ibm,connector-type"** property, there is assumed to be only one connector for the device (default connector) and this property should contain only one string. If multiple wrap plugs may be used with the same connector, their part numbers shall be represented in the same string, separated by commas.

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

"ibm,pci-config-space-type"

property name: Indicates if the platform supports access to an extended configuration address space from the PHB up to and including this node.

0 = Platform supports only an eight bit register number for configuration address space accesses.

1 = Platform supports a twelve bit register number for configuration address space accesses.

This property may be provided in all **PHB** nodes and their children.

Note: The absence of this property implies the platform supports only an eight bit register number for configuration address space accesses.

"ibm,reserved-explanation"

property name indicates why this PHB's **"status"** property contains the value of "reserved" or "reserved-uninitialized".

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

The property value, when present, can have the values specified in Table 246, ""ibm,reserved-explanation" Values," on page 705.

Table 246. **"ibm,reserved-explanation"** Values

Value	Explanation
storage-system-io	Reserved for storage system product use
pcix-over-pcie	PCIe device is abstracted as a PCIx device in the device tree for legacy compatibility

"ibm,pe-total-#msi"

property name defines the maximum number of Message Signaled Interrupts (MSI plus MSI-X) that are available to the PE below this device tree node. This number only indicates the number of available interrupts, not the number assigned. The number assigned for an IOA may be obtained by Function 0 (Query only) of the *ibm,change-msi* RTAS call.

prop-encoded-array: Maximum number of interrupts encoded as with **encode-int**.

"ibm,ehci-boot-supported"

property name: indicates if this IOA function for USB 2.0 (EHCI) supports devices beneath it to be used for boot.

prop-encoded-array: None.

"ibm,pe-reset-is-flr"

property name: The presence of this property in the PCI Express function's OF Device Tree node indicates that the platform will use the Function Level Reset (FLR) of the function to reset the function when the *ibm,set-slot-reset* RTAS call is used to reset the PE, and not the PCI Express Hot Reset.

prop-encoded-array: None.

"ibm,ddw-applicable"

property name: The Dynamic DMA Windows option RTAS calls may be used against the PE below this node.

prop-encoded-array: A list of three integers encoded as with **encode-int**.

This property may be provided in all PHB nodes or bridge nodes that are the PHB's children. Separate properties must exist for each PE that can participate in the DDW option (exists in the node above the PE). The existence of this property in any node, indicates that the platform supports the Dynamic DMA Windows option for the platform and for the PE below that node. Lack of this property in the bridge node above a PE indicates that the DDW option RTAS calls are not applicable to that PE. The values in the property are defined as follows:

The first integer represents the token to be used for the *ibm,query-pe-dma-window* RTAS call.

The second integer represents the token to be used for the *ibm,create-pe-dma-window* RTAS call.

The third integer represents the token to be used for the *ibm,remove-pe-dma-window* RTAS call.

"ibm,ddw-extensions"

property name: Extensions to the Dynamic DMA Windows option RTAS calls may be used against the PE below this node.

prop-encoded-array: A list of integers encoded as with *encode-int*.

This property may be provided in all PHB nodes or bridge nodes that are the PHB's children. Separate properties shall exist for each PE that can participate in the extensions to the DDW option (exists in the node above the PE).

The existence of this property in any node, indicates that the platform supports the extensions to the Dynamic DMA Windows option for the platform and for the PE below that node. Lack of this property in the bridge node above a PE indicates that the extensions of the DDW option RTAS calls are not applicable to that PE. This property is designed to be extended in the future by adding integers to the end of the list, reading software should be prepared to handle earlier versions of the property that will have a short list as well as ignore longer lists from later versions than it was designed to handle. The values in the property are defined as follows:

The first integer represents the number of extensions implemented. Subsequent integers represent values associated with each extension such as a token for an additional RTAS call or an architectural level of an extended interface. The value of one indicates that only a single extension is implemented as specified by the second integer in the list. Table 128, “DDW Option Extensions,” on page 263 provides the definition of the subsequent integers as defined for the LoPAPR level of the DDW option.

`ibm,h-get-dma-xlates-supported`

property name: to identify those PHBs for which H_GET_DMA_XLATES is supported on all child LIOBNs.

prop-encoded-array: <none>

`ibm,h-get-dma-xlates-limited-supported`

property name: to identify those PHBs for which H_GET_DMA_XLATES_LIMITED is supported on all child LIOBNs.

prop-endoded-array: <none>

B.6.5.1.1.2 LPAR Option Properties

`“ibm,dma-window”`

property name to define the bus address window children of this bridge may use for dma.

prop-encoded-array: One (*logical-bus-number, phys*, *size*) triple where the logical bus number (LIOBN) is a one cell cookie representing the unique range of TCE entries assigned to this bridge encoded as with **`encode-int`**, the number of cells for *phys* corresponds to the node’s **`“ibm,#dma-address-cells”`** value while the number of cells for *size* corresponds to the **`“ibm,#dma-size-cells”`** for this node.

Implementation Note: Platforms that support PHB level granularity of IO assignment to partitions place the **`“ibm,dma-window”`** property in the PHB node, while platforms that support slot level granularity place the **`“ibm,dma-window”`** property in the bridge node that creates the per slot bus isolation.

Note: The first element of the ibm,dma-window triple (the LIOBN) is used as a parameter to firmware DMA setup routines to identify the specific I/O address space (TCE table) to be referenced.

`“ibm,is-vf”`

property name to define that the node represents an I/O Virtualized instance of an I/O adapter.

prop-encoded-array: A one cell value that represents the LoPAPR architectural level of the virtualization:

Table 247. “`ibm,is-vf`” Values

Value:	Description:
0	Not used
1	Per LoPAPR

Table 247. "ibm,is-vf" Values

Value:	Description:
All others	Reserved

B.6.6 Memory Node

This section defines the LoPAPR modifications to the OF /**memory** node. In LoPAPR, the memory allocated to an OS image may be divided into a number of allocation units called "regions" or "Logical Memory Blocks (LMB). An OS image may be dynamically allocated additional regions or may be asked to release regions. Each LMB is either represented in the device tree by its own /**memory** node or by an entry in /**ibm,dynamic-reconfiguration-memory** nodes (see Section B.6.6.2, "ibm,dynamic-reconfiguration-memory," on page 708). The /**memory** node that refers to the storage starting at real address zero (**"reg"** property starting at the value zero) always remains allocated to an OS image. The client program is initially loaded into this storage, called the RMA, that is represented by the first value of the **"reg"** property of this first /**memory** node. Additional storage regions may each be represented by their own /**memory** node that includes dynamic reconfiguration (DR) properties or by an entry in /**ibm,dynamic-reconfiguration-memory** nodes.

To support dynamic addition and removal of regions, the / node contains the properties: *ibm,drc-types* (MEM), *ibm,drc-indexes ibm,drc-names* and *ibm,drc-power-domains* (-1's). These properties have entries for the maximum number of dynamically reconfigurable regions that the platform supports for the specific OS image.

B.6.6.1 Properties of the memory Node

In addition to the standard properties defined for the **/memory** node, the following are required for each node representing a dynamically allocable memory region. Platforms that support the dynamic reconfiguration of memory regions represent each such logical memory block with its own **/memory** node. Any new memory granted to an OS image is done so with a new **/memory** node, and OS images may free memory only in full blocks represented by one of its currently held **/memory** nodes.

The value of **"#address-cells"** for this node is 1.

The value of **"#size-cells"** for this node is 0 because the children of this node do not consume any physical address space.

The **"ibm,my-drc-index"** property as defined in Section B.6.1, "Properties for Dynamic Reconfiguration," on page 670.

"ibm,preservable"

property name that denotes the platform's ability to preserve the contents of the storage represented by this node.

prop-encoded-array: A integer encoded as with **encode-int** that represents the ability of the platform to preserve the contents of the storage.

All non-negative values represents the expected length of time, in minutes, that the platform can sustain the state of the storage. A value of 0 indicates the storage is not preservable and the client program may not register this storage for preservation, this is the assumed state if the **"preservable"** property is not present. The largest positive number represents an indefinite retention time as provided by such technologies as flash storage.

Negative values indicate that the storage is preservable as long as external power is maintained, perhaps by an external battery not directly integrated into the platform.

"ibm,preserved"

property name that denotes the preservation state of the contents of the storage represented by this node.

prop-encoded-array: An integer encoded as with **encode-int** that represents the preservation state of the storage. The defined states are:

0= The storage was not registered for preservation and thus not preserved. This is the assumed state if the **"preserved"** property is not present. This is also the state if the platform has lost knowledge of the preservation registration state of the storage.

1= The storage was registered for preservation and is has been preserved since the client program last modified it.

2= The storage was registered for preservation, however, the contents have not been preserved.

"ibm,expected#pages"

property name that denotes the number of pages that the client program is expected to use to virtually map the LMB represented by this node.

prop-encoded-array: An integer encoded as with **encode-int** that represents the log base 2 of the expected number of virtual pages that the client program will use to map the LMB represented by this node.

"ibm,no-h-migrate-dma"

property name that designates that the memory in the **memory** node in which this property resides cannot have the H_MIGRATE_DMA hcall() used against it.

prop-encoded-value: <none> this is a name only property.

B.6.6.2 ibm,dynamic-reconfiguration-memory

This device tree node defines an alternative means to represent a number of dynamically-reconfigurable logical memory blocks (LMBs). This node must only be generated by OF when instructed to do so by the client program in the ELF header. All memory which is not subject to dynamic reconfiguration (such as the RMA) is represented in **/memory** node(s).

This node is a child of root. This node does not have a unit address or **"reg"** property.

The following properties are defined.

"ibm,lmb-size"

property name that defines the size of each dynamically reconfigurable LMB.

prop-encoded-array: An integer encoded as with **encode-phys** that represents the size in bytes of each LMB.

"ibm,associativity-lookup-arrays"

property name that defines a lookup array in which to find the *ibm,associativity-array* property value for the LMBs.

prop-encoded-array: The number M of associativity lists encoded as with **encode-int**, the number N of entries per associativity list encoded as with **encode-int**, followed by M associativity lists each of length N integers encoded as with **encode-int**.

This property is used to duplicate the function of the **"ibm,associativity"** property in a **/memory** node. Each "assigned" LMB represented has an index valued between 0 and M-1 which is used as in index into this table to select which associativity list to use for the LMB. "unassigned" LMBs are place holders for potential DLPAR additions, for which the associativity list index is meaningless and is given the reserved value of -1. This static property, need only contain values relevant for the LMBs presented in the **"ibm,dynamicreconfiguration-memory"** node; for a dynamic LPAR addition of a new LMB, the device tree fragment reported by the *ibm,configure-connector* RTAS function is a /memory node, with the inclusion of the **"ibm,associativity"** device tree property defined in Section B.6.2.2, "Properties of the Children of Root," on page 679.

"ibm,dynamic-memory"

property name that defines memory subject to dynamic reconfiguration.

prop-encoded-array: The number N of LMB list entries defined at boot time, encoded as with **encode-int**, followed by N LMB list entries.

An LMB list entry consists of the following elements. There is one LMB list entry per LMB represented.

Logical address of the start of the LMB, encodes as with **encode-phys**. This corresponds to the first words in the **"reg"** property in a **/memory** device tree node.

DRC index of the LMB, encoded as with **encode-int**. This corresponds to the **"ibm,my-drc-index"** property in a **/memory** device tree node.

Four (4) bytes reserved for future expansion of flag.

Associativity list index for the LMB, encoded as with **encode-int**. This is used as an index into **"ibm,associativity-lookup-arrays"** property defined above to retrieve the associativity list for the LMB. The associativity list corresponds to the **"ibm,associativity"** property in a **/memory** device tree node.

A flags word, encoded as with **encode-int**. This word represents 32 boolean flags. As of this definition, flag bits are defined to correspond to the **"ibm,preservable"** and **"ibm,preserved"** properties in a **/memory** device tree node. This definition allows for additional flags to be added in the future.

The following bits in the "flags word" above are defined.

Table 248. Flag Word

Name	Bit Position	Description
preserved	0x00000001	If b'0', corresponds to the **"ibm,preserved"** property having a zero value. If b'1', corresponds to the **"ibm,preserved"** property having a non-zero value, and the preserved_state bit below indicates the state of preservation.
preservable	0x00000002	If b'0', corresponds to the **"ibm,preservable"** property having a zero value. If b'1', corresponds to the **"ibm,preservable"** property having a non-zero value.
preserved_state	0x00000004	If b'0', corresponds to the **"ibm,preserved"** property having a 0x1 value. If b'1', corresponds to the **"ibm,preserved"** property having a 0x2 value (and, in the old binding, the LMB having a status of "fail").
assigned	0x00000008	If b'1', this LMB is assigned to the partition as of boot time. If b'0', this LMB is not assigned to the partition as of boot time.
No H_MIGRATE_DMA	0x00000010	If b'0', corresponds to non-existence of the **"ibm,no-h-migrate-dma"** in the **memory** node. If b'1', corresponds to existence of the **"ibm,no-h-migrate-dma"** in the **memory** node.

Table 248. Flag Word *(Continued)*

DRC invalid	0x00000020	If b'0', the DRC field of **"ibm,dynamic-memory"** property is valid. If b'1', the DRC field of **"ibm,dynamic-memory"** property is invalid.
Associativity Index	0x00000040	If b'0', the Associativity List Index field of **"ibm,dynamic-memory"** property is valid. If b'1', the Associativity List Index field of **"ibm,dynamic-memory"** property is invalid.
Reserved Memory	0x00000080	If b'0', corresponds to the **"status"** property having a value of "okay". If b'1', corresponds to the **"status"** property having a value of "reserved".

"ibm,memory-flags-mask"

property name that defined which flags in the "flags word" above are defined in this version of this architecture.

prop-encoded-array: An integer encoded as with **encode-int** with all flag bits recognized by this version of this architecture having a b'1' value. For this version, the value will be 0x000000FF.

"ibm,memory-preservation-time"

property name that defined the time value that would appear in the **"ibm,preservable"** property in the old bindings for a preservable LMB.

prop-encoded-array: An integer value encoded as with **encode-int** that represents the expected length of time, in minutes, that the platform can sustain the state of power for a preservable LMB. The largest positive number represents an indefinite retention time as provided by such technologies as flash storage. A value zero indicates that no memory will be marked as preservable.

B.6.7 Memory Controller Nodes

This section describes **memory-controller** nodes and their properties. NUMA configurations, have multiple **memory-controller** nodes in the device tree one for each Central Electronics Complex (CEC). In dynamic reconfiguration NUMA environments, these **/memory-controller** nodes are subject to standard LoPAPR dynamic reconfiguration operations and contain standard LoPAPR dynamic reconfiguration properties.

B.6.7.1 Memory Controller Node Properties

No nodes of type **memory-controller** shall be defined anywhere in the device tree when the platform fully abstracts the memory controller and the OS has no access to the memory controller (typically when running in a partition). Otherwise, one or more nodes of type **memory-controller** shall be defined as a child of "/" (the root) and shall not have a **"ranges"** property. The following properties shall apply to each of these nodes. If the platform does not abstract the functions of a platform's multiple memory controllers via firmware (such as RTAS) then the platform shall include a node of type **memory-controller** for each Memory Controller in the platform.

A Memory Controller can also have the **"used-by-rtas"** property (see Section B.6.10.2, "Miscellaneous Node Properties," on page 717), if it has functions abstracted by RTAS.

"device_type" S

Standard *property name* that denotes a Memory Controller node.

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

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

"reg" S

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

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

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

"model" S

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

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

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

"external-control" S

property name indicates this Memory Controller's ability to support the PA external control facility.

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

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

"error-checking" S

Standard *property name* defines the error checking capability of the node.

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

The value of **"#address-cells"** for this node is 1.

The value of **"#size-cells"** for this node is 0 because the children of this node do not consume any physical address space.

B.6.8 IBM,memory-module Nodes

Memory packaged on DIMMs or DIMM like modules are represented in the device tree with **IBM,memory-module** nodes. These nodes represent physical packages, these packages do not necessarily map directly to a memory address range.

No nodes of type **IBM,memory-module** shall be defined anywhere in the device tree when the platform supports dynamic VPD via the RTAS *ibm,get-vpd* service. Instead the VPD that would normally be reported via the **"ibm,vpd"** property in these nodes shall be reported by *ibm,get-vpd*.

B.6.8.1 Properties for Memory Modules

Memory modules appear as children of the **memory** node or, for platforms supporting memory DR operations (either logical or physical), the **memory-controller** node of the device tree. This section defines properties for the **IBM,memory-module** nodes and additional properties for the **memory** and **memory-controller** nodes.

B.6.8.2 `IBM,memory-module` Node Properties

An **`IBM,memory-module`** node is a child of the **`memory`** node or, for platforms supporting memory DR operations (either logical or physical), the **`memory-controller`** node.

The **`"name"`** of the node is "IBM,memory-module"

The **`"device_type"`** of the node is "IBM,memory-module"

The **`"reg"`** standard property for an **`IBM,memory-module`** node is its memory module number which is an arbitrary OF selected enumeration.

The **`"ibm,size"`** property for an **`IBM,memory-module`** node is an integer which is less than 4GB and which by itself indicates the size of the memory module, in bytes, if the memory module is smaller than 4GB and if `"status"` is `"okay"` or `"fail"`.

If the memory module is larger than or equal to 4GB in size, then the **`"ibm,size-upper"`** property for an **`IBM,memory-module`** node is present in addition to the **`"ibm,size"`** property. This property is an integer which is multiplied times 4GB and then added to the value of the **`"ibm,size"`** property to get the size of the module, in bytes. The property **`"ibm,size-upper"`** is not required if the memory module size is less than 4GB.

The **`"status"`** standard property for an **`IBM,memory-module`** node may have one of the following string values:

`"okay"` for a good memory module

`"fail"` for a bad memory module

`"fail-no-matched-pair"` for a missing memory module if one of a pair is missing

`"fail-unsupport"` for an unsupported memory module

`"fail-partial"` for a bad memory module where part of the memory on the module is bad and has not been configured and part of the memory is good and has been configured.

`"fail-excess-memory"` for `"okay"` memory modules that are not configured because they exceed the system memory addressability of the platform.

`"disabled"` for a memory module that has been manually deconfigured.

`"ibm,mem-banks"`

property name defines the number of memory banks contained within the memory module.

prop-encoded-array: an integer, encoded as with **`encode-int`**, which describes whether this is a 1, 2, or 4-bank module, with a corresponding value of 1, 2 or 4 and so on to match the number of banks in the physical device.

`"ibm,mem-type"`

property name defines the memory module type.

prop-encoded-array: a string, encoded as with **`encode-string`**, that describes the type of module, with values of `"FP"` (Fast Page), `"EDO"` (Extended Data Out), or `"SDRAM"` (Synchronous DRAM).

`"ibm,mem-err-det"`

property name defines the type of error detection mechanism supported by the module

prop-encoded-array: a string, encoded as with **`encode-string`**, with values of `"none"`, `"parity"`, or `"ECC"`.

`"ibm,mem-speed"`

property name defines the access or clock speed supported by the module, in picoseconds

prop-encoded-array: an integer, encoded as with **encode-int**, which describes the access or clock speed supported by the module, in picoseconds.

B.6.9 Interrupt Controller Nodes

This section describes the properties for the LoPAPR interrupt controller node. If an interrupt controller node includes the **"used-by-rtas"** property, then the platform includes firmware code for accessing the interrupt controller.

For LSIs, the platform shall adhere to the *Open Firmware: Recommended Practice - Interrupt Mapping, Version 1.0* [9] interrupt structure OF representation.

B.6.9.1 PowerPC External Interrupt Controller Nodes

This section describes the properties for the PowerPC External Interrupt Controller nodes. PowerPC interrupt controllers are normally packaged inside other system chips, however, they are logically represented in the device tree by two or more independent interrupt controller nodes. Each node reports either the interrupt source layer resources that are housed in a single Bus Unit Controller (BUC) e.g. host bridge, or logical equivalent, or a subset of the resources associated with the platform's interrupt presentation layer. The node per BUC and presentation layer subset divisions provides a foundation for dynamic reconfiguration.

At a dynamic reconfiguration event, such as adding an IO drawer, or removing a processor, the interrupt controller nodes associated with the added or removed hardware will also be added or removed. Therefore. platforms should report, in individual nodes, each interrupt controller that occupies a separate physical package. And OSs should expect a fine granularity of interrupt controller reporting.

"ibm,interrupt-domain"

property name that denotes a PowerPC External Interrupt Domain

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

B.6.9.1.1 PowerPC External Interrupt Presentation Controller Node Properties

The following properties apply to this node.

"name" S

Standard *property name* that denotes a PowerPC External Interrupt Controller.

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

The value of this string shall be "interrupt-controller".

"device_type" S

Standard *property name* that indicates an Interrupt Controller.

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

The value of this property shall be "PowerPC-External-Interrupt-Presentation".

"reg" S

Standard *property name* defines the base physical address(s) and size(s) of this PowerPC External Interrupt Presentation layer's registers.

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

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

The entries shall represent the base address of a single set of PowerPC External Interrupt Presentation Layer Registers of the Interrupt Management Area. There shall be one entry for each interrupt server queue supported by this unit. The order of the entries shall correspond to the entries in the **"ibm,interrupt-server-ranges"** property described below.

"compatible" S

Standard *property name* to define alternate **"name"** property values.

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

The property value shall include "IBM,ppc-xicp".

"ibm,interrupt-buid-size"

property name: Defines the number of bits implemented in the concatenation of the BUID fields.

prop-encoded-value: An integer in the range of 9 to 20 encoded as with **encode-int**.

As platforms grow in size so as to require use of larger BUIDs (values of the **"ibm,interrupt-buid-size"** property greater than 9) the platform engineers need to interlock with their OS providers to ensure support.

"ibm,interrupt-server-ranges"

Property name that defines the interrupt server number(s) and range(s) handled by this unit.

prop-encoded-array: List of (*server#, range*) specifications.

Server# is encoded as with **encode-int** in the range of 0 - 2^(the largest value of the **"ibm,interrupt-server#-size"** property contained in the device tree).

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

The first entry in this list shall contain the *server#* associated with the first **"reg"** property entry. The *server#* corresponds to a value of a processor's **"ibm,ppc-interrupt-server#s"** property. The range shall be the number of contiguous *server#*s supported by the unit (this also corresponds to the number of **"reg"** entries).

"interrupt-controller" S

Standard *property name* to indicate an interrupt (sub-)tree root.

prop-encoded-array: <none> The presence of this property indicates that this node represents an interrupt controller.

"model" S

Standard *property name* indicating this unit's manufacturer, part number, and revision level.

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

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

B.6.9.1.2 PowerPC External Interrupt Source Controller Node Properties

Interrupt source controller resources as represented by **"interrupt-ranges"**, **"#interrupt-cells"**, and **"ibm,interrupt-server#-size"** properties may be reported in stand-alone interrupt source controller nodes or in other logical equivalent nodes which contain the **"interrupt-controller"** property. The following properties apply to these nodes.

"name" S

Standard *property name* that denotes a PowerPC External Interrupt Controller.

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

The value of this string shall be "interrupt-controller".

"device_type" S

Standard *property name* that indicates the specific flavor of Interrupt Source Controller.

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

The value of this property shall be one of the following:

"PowerPC-LSI-Source".
For Level Sensitive Interrupt source controllers.

"PowerPC-MSI-Source".
For Message Sensitive Interrupt source controllers such as used with PCI MSI.

"reg" S

Standard *property name* defines the base physical address(s) and size(s) of this PowerPC External Interrupt Source if any.

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

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

If the **"device-type"** of the interrupt source controller is "PowerPC-MSI-Source", then the last **"reg"** entry shall correspond to the interrupt controller's 4 byte Message Interrupt Input Port.

"compatible" S

Standard *property name* to define alternate **"name"** property values.

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

The property value shall include "ibm,ppc-xics".

"interrupt-ranges" S

Standard *property name* that defines the interrupt number(s) and range(s) handled by this unit.

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

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

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

The first entry in this list shall contain the *int-number* associated with the first **"reg"** property entry. The *int-number* is the value representing the interrupt source as would appear in the PowerPC External Interrupt Architecture XISR. The range shall be the number of sequential interrupt numbers which this unit can generate.

"interrupt-controller" S

Standard *property name* to indicate an interrupt (sub-)tree root.

prop-encoded-array: <none> The presence of this property indicates that this node represents an interrupt controller.

"model" S

Standard *property name* indicating this unit's manufacturer, part number, and revision level.

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

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

"#interrupt-cells" S

Standard *property name* to define the number of cells in an *interrupt-specifier* within an interrupt domain.

prop-encoded-array: An integer, encoded as with **encode-int**, that denotes the number of cells required to represent an interrupt specifier in its child nodes.

The value of this property for the PowerPC External Interrupt option shall be 2. Thus all interrupt specifiers (as used in the standard **"interrupts"** property) shall consist of two cells, each containing an integer encoded as with **encode-int**. The first integer represents the interrupt number the second integer is the trigger code: 0 for edge triggered, 1 for level triggered.

"ibm,interrupt-server#-size"

property name: Defines the number of bits implemented in the concatenation of the server#extension and server# fields.

prop-encoded-value: An integer in the range of 8 to 24 encoded as with **encode-int**.

As platforms grow in size so as to require use of the server#extension field (values of the **"ibm,interrupt-server#-size"** property greater than 8) the platform engineers need to interlock with their OS providers to ensure support.

B.6.10 Additional Node Properties

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

B.6.10.1 Interrupt Properties

The properties in this section shall be implemented for any device that can present an interrupt for an LoPAPR platform implementation. The platform shall adhere to the *Open Firmware: Recommended Practice - Interrupt Mapping, Version 1.0* [9] definition for the interrupt structure.

B.6.10.2 Miscellaneous Node Properties

This section defines properties which support devices, adapter and buses with geographical information. These properties shall be present for an LoPAPR platform.

"built-in" S

Standard *property name*: Any device that connects to an industry standard I/O expansion bus attached through a non-standard connector.

prop-encoded-string: <none>.

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

"used-by-rtas" S

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

prop-encoded-int: Presence of property indicates a device may have an I/O or resource conflict with a RTAS Function Call.

The use of the **"slot-names"** property defined below is deprecated in favor of the **"ibm,loc-code"** property.

"slot-names" S

property name: Describes external labeling of adapter/device connectors.

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

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

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

"ibm,loc-code" S

property name to provide location code(s) for the Field Replacable Unit.

prop-encoded-array: an arbitrary number of strings, encoded as with **encode-string**.

"ibm,vpd"

property name to provide Vital Product Data (VPD) information as defined in Section 12.4, "Vital Product Data," on page 341.

prop-encoded-array: the concatenation, with **encode+**, of one or more pairs of elements, the first element of each pair being an integer (representing the length of the second element) encoded as with **encode-int**, and the second element of each pair being a series of bytes (the VPD data) encoded as with **encode-bytes**.

"ibm,loc-code-map"

prop-name to identify that the interface may have child nodes, which may or may not be present in the device tree, that have a physical location code based on their unit-address.

prop-encoded-array: A list of pairs (unit-address, location-code). The unit-address is the child device node's **"reg"** property string-encoded according to the parent node's architecture and encoded as with encode-string. The location-code is the child device node's **"ibm,loc-code"** property encoded as with encode-string.

If a child device under this node has a matching unit-address, the location code corresponds to the physical location of that child device.

B.6.11 `/aliases` Node

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

An implementation of OF for an LoPAPR platform shall provide the following aliases as properties of the **`aliases`** node, if the corresponding device exists:

`"disk"` S

property name indicating the device path of the factory default disk that is the preferred boot disk[1] for the platform.

`"tape"` S

property name indicating the device path of the factory default tape.

`"cdrom"` S

property name indicating the device path of the factory default CDROM.

`"keyboard"` S

property name indicating the device path to the keyboard to be used for the User Interface.

`"mouse"` S

property name indicating the device path to the mouse to be used for the User Interface.

`"screen"` S

property name indicating the device path to the screen to be used for the User Interface.

`"pc-keyboard"` S

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

`"pc-mouse"` S

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

`"adb-keyboard"` S

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

`"adb-mouse"` S

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

`"scsi"` S

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

1.**Implementation Note:** The preferred boot disk should be the disk that results in the fastest boot time. Implementations might automatically spin up a disk at system power on and provide mechanisms for firmware to report that disk in this property.

"com1" S

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

"com2" S

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

"scca" S

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

"sccb" S

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

"floppy" S

property name indicating the device path of the factory default floppy drive.

"net" S

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

"rtc" S

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

"nvram" S

property name indicating the device path of the factory default NVRAM.

B.6.12 `/event-sources` Node

The **`/event-sources`** node describes the possible RTAS Error and Event Classes for interrupts. The **`/event-sources`** node shall be defined to be a child of the root device tree node if the platform supports any event interrupts. The following properties shall be defined for this node:

"name" S

Standard *property name* that denotes the Event Sources.

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

The value of this string shall be `"event-sources"`.

When events are reported as virtual interrupts there shall be a node of **`device_type "PowerPC-External-Interrupt-Presentation"`** from which the virtual interrupt source BUID size can be obtained. Also the **`event-sources`** node represents the interrupt source node for virtual event interrupts and thus the following properties shall be defined for this node:

"interrupt-controller" S

Standard *property name*: to indicate the events interrupt tree root.

prop-encoded-array: <none> The presence of this property indicates that this node represents a source of virtual interrupts. Encoded with **`encode-null`**.

"#interrupt-cells" S

Refer to the definition of the **"#interrupt-cells"** property for nodes of **device_type "PowerPC-LSI-Source"** for information about the definition of this property for virtual event interrupts.

"interrupt-ranges" S

Refer to the definition of the **"interrupt-ranges"** property for nodes of device_type **"PowerPC-LSI-Source"** for information about the definition of this property for virtual event interrupts.

Children of **/event-sources** present the interrupt specifiers associated with the reporting of platform events. LoPAPR platforms have historically implied the default value of **"#interrupt-cells"** of 1 to report the associated interrupt specifiers without the interrupt trigger specifier. However, all new designs shall present interrupt specifiers with explicit trigger level values.

B.6.12.1 Child nodes of the Event Sources Node

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

Children of the **/event-sources** node specify the interrupt specifiers associated with the reporting of platform events. Interrupt designs shall use the 1275 standard **"interrupts"** property as configured to report the interrupt specifier for the platforms PowerPC interrupt controller. The interrupt specifiers if the **"interrupts"** property indicates one or more interrupt source numbers that are used to report event conditions.

B.6.12.1.1 internal-errors

The presence of the node indicates that all or some of the function has been implemented and will be reported using an interrupt.

"name" S

Standard *property name* that denotes the internal error's events.

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

The value of this string shall be "internal-error".

B.6.12.1.2 epow-events

The presence of the node indicates that all or some of the function has been implemented and will be reported using an interrupt.

"name" S

Standard *property name* that denotes the EPOW events.

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

The value of this string shall be "epow-events".

B.6.12.1.3 ibm,io-events

The presence of the node indicates that all or some of the function has been implemented and will be reported using an interrupt.

"name" S

Standard *property name* that denotes the I/O sub-system events.

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

The value of this string shall be "ibm,io-events".

B.6.13 /reserved Node

This section defines a reserved node which shall have a **"reg"** property which allocates addresses (on the bus of which it is a child) which is intended to be a place to identify hardware registers that do not otherwise belong to a recognized device.

"name" S

Standard *property name* that denotes reserved addresses that do not belong to a recognized device.

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

The value of this string shall be "reserved".

"device_type" S

Standard *property name* that indicates the device type.

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

The value of this property shall be "reserved".

"reg" S

Standard *property name* defines a hardware register address and range of addresses not intended for OS (OS) use.

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

Phys-addr is a (phys.lo ... phys.hi) sequence equal to **#address-cells**, encoded as with **encode-phys**. *size* is a sequence equal to **#size-cells** encoded as with **encode-size**.

The first entry in this list shall be a hardware register address (*phys-addr*) and a range of hardware addresses (*size*) that is not intended for OS usage. Successive entries in this list shall be additional hardware addresses not intended for OS usage.

B.6.14 /chosen Node

This section lists additional properties as required under the **/chosen** node with the following text in a manner that is consistent with *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], Section 3.5.

"nvram" S

Standard *property name* that defines the package *Ihandle* for CHRP NVRAM.

prop-encoded-array: an integer, as encoded with **encode-int**, that is the package *Ihandle* the CHRP NVRAM.

Note: The nvram Node identified in the /chosen Node shall support a size method as specified in *Open Firmware Recommended Practice: Device Support Extensions* [5], Section 7.2. The size method will return a value that is the total platform NVRAM size.

"ibm,rpa-client-config"

property name that defines the processed fields of the client program's IBM,RPA-Client-Config ELF note section.

prop-encoded-array: an array of integers encoded as with **encode-int**, that consist of the fields of the note section that the firmware processed prior to loading the client program.

"ibm,architecture-vec-5"

property name: that presents the values of the option vector #5 negotiated by the **ibm,client architecture-support** method. Presence of this property signifies that the client program load module invoked the **ibm,client architecture-support** method.

prop-encoded-array: An array of bytes having the format of the fifth option vector from Table 244, "ibm,architecture.vec option vectors," on page 681 representing the value chosen by the **ibm,client architecture-support** method.

"ibm,client-architecture-support-reboot"

property name: that indicates that one or more reboots have occurred in this boot sequence in order to adjust the platform settings to match the specification in the **"ibm,client-architecture-support"** open firmware method or the IBM,RPA-Client-Config ELF header note. Note this property is not included for the first boot in a sequence.

prop-encoded-array: encoded as with **encode-int** that specifies the number of reboots that have occurred in this boot sequence in order to adjust the platform settings to match the specification in the **"ibm,client-architecture-support"** open firmware method or the IBM,RPA-Client-Config ELF header note.

B.6.15 /vdevice Node

The node of type vdevice is a child of the root node. It is only present in trees that also include the **"ibm,hypertas_functions"** property. It, and its children represent the virtualized devices that are implemented by the platform firmware. Virtualized devices do not surface to a client program a direct hardware interface. They do not appear to take up space in the client program's address map. Standard property names associated with the **/vdevice** node have special values as specified below.

"#address-cells" S

Standard *property name* encoded as with **encode-int** that specifies the number of cells required to represent a child bus address. Shall have the value of 1.

"#size-cells" S

Standard *property name* encoded as with **encode-int** that specifies the number of cells required to encode the size field of a child's reg property. Shall have the value of 0 indicating that no child node may actually take physical address space.

"name" S

Standard *property name* string encoded as with **encode-string** that defines the name of node. The value shall be the string "vdevice".

"device_type" S

Standard *property name* string encoded as with **encode-string** that defines the device type of the node. The value shall be the string "vdevice".

"ibm,max-virtual-dma-size"

Vendor unique *property name* indicating the maximum size virtual dma transfer size supported by the platform

prop-encoded-array: a single integer encoded as with **encode-int**.

"ibm,migration-control"

property name that indicates when platform firmware supports the ability for an I/O server partition to delay the migration of a partition to a different server in order to let any in progress I/O to be completed. Specifically, this property indicates that the DISABLE_MIGRATION and ENABLE_MIGRATION subfunctions of the H_VIOCTL hypervisor call are supported.

prop-encoded-array: None, this is a name only property.

"ibm,reserved-virtual-addresses"

Vendor unique *property name* indicating ranges of the client program virtual address space that are reserved for platform use.

prop-encoded-array: one or more pairs of abbreviated-virtual-address, virtual-address-length specifications representing the origin and length respectively of a reserved virtual address range.

abbreviated-virtual-address: Consists of two integers encoded as with *encode-int* representing the high order and low order 32 bits respectively of the 64 bit abbreviated virtual address. The full virtual address is the abbreviated-virtual-address concatenated with 3 low order bytes of 0x00.

virtual-address-length: Consists of a single integer encoded as with *encode-int* representing the number of consecutive 4K pages contained within the range.

B.6.15.1 Children of the `/vdevice` Node

The children of the **/vdevice** node represent the individual virtual devices.

Children of the **/vdevice** node that support dma operations contain a the **"ibm,my-dma-window"** property as defined below:

"ibm,my-dma-window"

property name that defines the bus address window(s) that this IOA may use for its dma.

prop-encoded-array: One or more (*logical-I/O-bus-number, phys*, *size*) triple(s) where the logical bus number is a one cell cookie representing the unique range of TCE entries assigned to this IOA encoded as with **encode-int**, the number of cells for *phys* corresponds to the node's **"ibm,#dma-address-cells"** value while the number of cells for *size* corresponds to the **"ibm,#dma-size-cells"** for this node. The first triple represents the TCE range available for mapping local memory, while the second triple, if it exists, is where remote memory mapped by remote partitions appears. The size field of the second triple shall be equal to the size field of the corresponding remote partition's first triple.

The **"ibm,my-dma-window"** property is the per device equivalent of the **"ibm,dma-window"** property found in nodes representing bus bridges.

Children of the **/vdevice** node share the ability to display unique capabilities as represented by the following properties.

"ibm,async-dma-required"

property name indicates that the virtual device requires the use of asynchronous virtual DMA interfaces (see *ISO-9660, Information processing -- Volume and file structure of CD-ROM for information interchange* [14] for definition of asynchronous virtual DMA interfaces).

prop-encoded-array: None, this is a name only property.

Children of the **/vdevice** node which act a a server to other virtual client devices, display the following property.

"ibm,vserver"

property name indicates that the virtual device is a server to virtual devices.

prop-encoded-array: None, this is a name only property.

"ibm,mac-address-filters"

property name specifies the number of non-broadcast multicast MAC filters supported by the implementation.

prop-encoded-array: an integer in the range of 0-255 encoded as with **encode-int**.

"ibm,trunk-adapter"

property name that indicates that the virtual device is a trunk adapter server to the logical LAN.

prop-encoded-array: None, this is a name only property.

"ibm,illan-options"

property name: The existence of this property is required when any of the ILLAN sub-options are implemented and indicates that the H_ILLAN_ATTRIBUTES hcall() is implemented, and that hcall() is then used to determine which ILLAN options are implemented.

prop-encoded-array: None, this is a name only property.

B.6.15.1.1 Virtual Teletype Device

The virtual teletype device allows communication through the platform's attached Hardware System Console. There is one such virtual device node for each virtual terminal enumerated by the **"ibm,termno"** property. The unit addresses of the virtual teletype devices shall correspond to the enumeration presented in the **"ibm,termno"** property. Such virtual terminals, as represented by the **"ibm,termno"** property, are intended for the use of the client program and shall not be marked **"used-by-rtas"**. Similarly they may be "chosen" as the default input and output device.

"name" S

Standard *property name* encoded as with **encode-string** that defines the device name. The value shall be the string "vty".

"reg" S

Standard *property name* to define a unit address for the node. One (*phys-addr, size*) pair. The *phys-addr* is the unit address of the device (corresponding to one of the virtual terminals enumerated by the **"ibm,termno"** property), and the *size* shall have a length of zero.

"device_type" S

Standard *property name* encoded as with **encode-string** to specify the device type. The value shall be the string "serial" indicating that the device emulates a serial terminal.

"compatible" S

Standard *property name* encoded as with **encode-string** to specify the device driver compatibility. The value shall be one of the strings specified in Table 249, "Virtual tty compatibility strings," on page 725.

Table 249. Virtual tty compatibility strings

Compatible property String Value	Comments
"hvterm1"	Standard client virtual tty protocol
"hvterm2"	Standard server virtual tty protocol
"hvterm-protocol"	Client virtual tty protocol extended for control of modems etc.

See Section 16.6, "Virtual Terminal (Vterm)," on page 582 for further detail on this virtual device.

B.6.15.1.2 Children of /vdevice node defined in other documents

Like children of the pci bus node, children of **/vdevice** may be defined by their own binding documents or via binding sections/tables in their device specifications. For example, the binding information for the LoPAPR Interpartition Logical LAN, Virtual SCSI, and Virtual Terminal can be found in the appropriate sections of this document. The virtualization of traditional physical devices repositions their associated device tree nodes to be children of **/vdevice**. Examples include NVRAM and Real Time Clock (RTC) devices which are defined by *Open Firmware Recommended Practice: Device Support Extensions* [5].

B.6.16 Barrier Synchronization Facility

This section describes the OF node that represents the optional Barrier Synchronization Register (BSR) facility. If the platform provides a BSR facility it provides the **ibm,bsr** node as a child of **/** (root). If the platform provides a client program with multiple independent facilities, it represents each such facility with a separate node. A given facility may have multiple representations through parallel windows. Each window of a given facility is represented by a separate **"reg"** property value. The following properties are the minimum required, optional support such as dynamic reconfiguration will add properties per requirements called out in the Section 13.5.2, "For All DR Options - OF Requirements," on page 362.

"name" S

Standard *property name* encoded as with **encode-string** that defines the device name. The value shall be the string "ibm,bsr" for legacy implementations and "ibm,bsr2" for POWER8 implementations and beyond.

"reg" S

Standard *property name* to define the addresses for the facility window(s).

prop-encoded-array: One or more (*phys-addr, size*) pair(s). The *phys-addr*, encoded as with **encode-phys,** is the starting address (4 K aligned) of the window. The *size*, encoded in the number of cells specified by **"#size-cells"** of the parent, is the length of the corresponding window.

"device_type" S

Standard *property name* encoded as with **encode-string** to specify the device type. The value shall be the string "ibm,bsr".

"compatible" S

Standard *property name* encoded as with **encode-string** to specify the device driver compatibility. The value shall be the string "ibm,bsr".

"ibm,#lock-bytes"

property name: Indicates the number of lock bytes per line of the BSR facility.

prop-encoded-array: One or more integers encoded as with **encode-int**. When the facility has multiple windows, as represented by multiple values of the **"reg"** property, then there is a corresponding number of integers, each integer representing the number of lock bytes per line of the corresponding window.

"ibm,lock-stride"

property name: Indicates the number of bytes between the beginning of lock lines in the BSR facility.

prop-encoded-array: One or more integers encoded as with **encode-int**. When the facility has multiple windows, as represented by multiple values of the **"reg"** property, then there is a corresponding number of integers, each integer representing the number of bytes to the beginning of the next lock line in the corresponding window.

B.6.17 Nodes of device_type "block" and "byte"

This section describes the OF nodes that provide access to storage devices in block or byte commands. This applies to such nodes with and without a **"reg"** property.

"ibm,write-supported"

property name: Indicates the driver supports write functionality and has been verified by IBM. The use of the write function without this property is discouraged.

prop-encoded-array: None, this is a name only property.

"ibm,16byte-cdb-supported"

property name: Indicates the driver supports using the 16 byte Command Descriptor Block format, which is needed to access above 2 TB on 512 byte block-sized media.

prop-encoded-array: None, this is a name only property.

B.6.18 /ibm,platform-facilities

The node of type **ibm,platform-facilities** is a child of the root node. It and its children represent the non-CPU platform computational facilities that are available. Platform facilities do not take up space in the client program's address map. Standard property names associated with the **/ibm,platformfacilities** node have special values as specified below.

"#address-cells" S

Standard *property name* encoded as with **encode-int** that specifies the number of cells required to represent a child bus address. Shall have the value of 1.

"#size-cells" S

Standard *property name* encoded as with **encode-int** that specifies the number of cells required to encode the size field of a child's reg property. Shall have the value of 0 indicating that no child node may actually take physical address space.

"name" S

Standard *property name* string encoded as with **encode-string** that defines the name of node. The value shall be the string **"ibm,platform-facilities"**.

"device_type" S

Standard *property name* string encoded as with **encode-string** that defines the device type of the node. The value shall be the string **"ibm,platform-facilities"**.

Some platform facilities configurations allow multiple facilities to share a common pool of interrupt server numbers. Individual operations specify which interrupt server number from the pool shall be used to signal completion of the operation. To represent such a configuration, the **/ibm,platformfacilities** node may either represent an interrupt source controller for its children or the interrupt source controller associated with the shared pool may be represented by a PowerPC External Interrupt Source Controller Node as an additional child node of the **/ibm,platform-facilities** node (See Section B.6.9.1.2, "PowerPC External Interrupt Source Controller Node Properties," on page 715). Additionally, the node representing the platform facilities Interrupt Source Controller shall contain the **"ibm,interrupt-pool"** property, and all platform facilities that share the common pool of interrupts shall contain the **"ibm,shared-interrupt-pool"** property.

"ibm,interrupt-pool"

property name: that indicates this interrupt controller provides a shared pool of interrupt source numbers.

property encoded array: single cell encoded as with **encode-int** that represents the type of shared interrupt pool being represented: Defined values are: 0 with all other values reserved.

"ibm,max-async-ops-per-processor"

property name: that indicates for the partition the allowed maximum number of outstanding operations for the platform facility based upon the number of processors currently allocated to the partition. The total allowable number of such operations outstanding across all partition processors is the product of the value of **"ibm,max-async-ops-per-processor"** and the number of nodes of type cpu in the current partition device tree.

property encoded array: single cell encoded as with **encode-int**

B.6.18.1 Children of the /ibm,platform-facilities Node

The children of the **/ibm,platform-facilities** node represent the individual platform facilities. Standard property names associated with the children of the **/ibm,platform-facilities** node have special values as specified below. Note the children of the **/ibm,platform-facilities** node shall contain the following standard properties with their standard definitions:

- **"compatible"**
- **"name"** The defined Values for the **"name"** property of children of **/ibm,platform-facilities** are (were # is the version number of the interface):
  - **ibm,random-v#** Random number generator
  - **ibm,compression-v#** Compression/Decompression engine
  - **ibm,sym-encryption-v#** Symmetric encryption/decryption engine
  - **ibm,asym-encryption-v#** Asymmetric encryption/decryption engine
- **"status"**

Optionally the children of the **/ibm,platform-facilities** node may contain as appropriate the following standard properties with their standard definitions:

- **"interrupts"**

Additionally the children of the **/ibm,platform-facilities** node may contain as appropriate the following unique properties:

"ibm,resource-id"

property name: that indicates the platform facility resource identification handle.

property encoded array: single cell encoded as with **encode-int**

"ibm,max-sync-cop"

property name: that indicates the maximum characteristics of the parameters for a synchronous call of the platform facility. These characteristics are represented as a series of integers encoded as with **encode-int** that may grow over time as platform facilities evolve. The absence of this property indicates that synchronous operations are not allowed for the given child.

property encoded array: a series of zero or more or more cells each encoded as with **encode-int.** The interpretation of the series of integers is unique per the value of the **"name"** property:

- For the Random number generator: NULL value indicating that all calls are synchronous
- For the compression/decompression engine: Two series of cells the first series of cells represents the maximums that can be synchronously compressed. The second series of cells represents the maximums that can be synchronously decompressed.
  - The first cell in each series contains the count of the number of data length, scatter/gather elements pairs that follow – each being of the form
    - One cell data byte length
    - One cell total number of scatter/gather elements
- For the symmetric encryption/decryption engine: the series of cells report for each function code (FC) and mode combination the maximum amount of data and scatter/gather list elements that can be processed with a given key length. Thus the array consists of 1-N sub sequences each of the form:
  - First cell contains the FC field
  - Second cell contains the mode field
  - Third cell contains the count of the number of key length, data length, scatter/gather elements triples that follow – each being of the form:
    - One cell key bit length
    - One cell data byte length
    - One cell total number of scatter/gather elements

"ibm,max-sg-len"

property name: that indicates the maximum byte length of a scatter/gather list for the platform facility.

property encoded array: single cell encoded as with **encode-int**

"ibm,shared-interrupt-pool"

property name: that provides an indirect pointer to the node representing the shared interrupt pool used by this facility.

property encoded array: the phandle of the node representing the PowerPC External Interrupt Source Controller that sources the interrupts of the shared interrupt pool used by this facility.

B.7 Symmetric Multi-Processors (SMP)

LoPAPR platforms can have Symmetric Multi-Processor (SMP) Configurations. In addition to the processor node properties defined in Appendix C, "PA Processor Binding," on page 753, a SMP Configuration will utilize the **/cpus** node as explained in Section B.7.1, "SMP Platform Device Tree Structure," on page 729

B.7.1 SMP Platform Device Tree Structure

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

The **/cpus node** shall have one child node of device type cpu for each processor. The *ihandle* of the "executing" processor shall be published in the **"cpu"** property of the **/chosen** node.

Note: The properties of a cpu device are already defined in Appendix C, "PA Processor Binding," on page 753. The only change for symmetric multiprocessor (SMP) systems is that there will be a cpu device node under the **/cpus** node for each individual processor. Other properties of the cpu devices shall conform with the requirements stated in Appendix C, "PA Processor Binding," on page 753.

B.7.2 SMP Properties

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

"slot-names" S

property name that describes platform labeling of plug-in cpu/processor card slots.

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

The integer portion of the property value is a bitmask of possible processors; for each add-in slot on the bus, the bit corresponding to that slot's ID number is set from least-significant to most-significant ID number. The number of following strings is the same as the number of slots; the first string gives the platform nomenclature for the slot with the smallest ID number, and so on. The CPU's **"slot-names-index"** property can be used as an index into the bitmask integer of this property. The absence of this property indicates that no slots are present.

"smp-enabled" S

property name that indicates a platform can be SMP enabled.

prop-encoded-array: <NULL>

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

B.7.2.1 Processor Node

The following properties are for a PA SMP environment. This SMP property will be under each **/cpu** Node.

"slot-names-index" S

property name: Identifies each cpu with a unique number.

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

The value of this integer is a platform unique number with a range from 0 to *n-1* for each CPU where *n* is the number of slots. This number is used to index into the **`"slot-names"`** property to identify the value of the string associated with the slot name.

B.8 Device Power Management Properties/Methods

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

B.8.1 System Node Properties

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

B.8.1.1 Properties assigned to the RTAS node

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

`"power-domains-tree"` S

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

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

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

`"power-domains-controllers"` S

Standard *property name* which defines the power domain controllers present on this platform.

prop-encoded-array: an array of integers, each encoded as with **`encode-int`**.

Each integer is the *phandle* of the device tree node that functions as the power domain controller for a domain. A single controller may serve as the control point for multiple domains (the architecture calls them power domain control points). Each device which serves as a controller encodes the **`"controls-power-domain"`** property.

`"power-domains-names"` S

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

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

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

`"platform-power-sources"` S

Standard *property name* defining the platform power sources.

prop-encoded-array: an array of integers, each encoded as with **`encode-int`**.

The array is structured as a number of tuples. Each of these tuples consists of the values source-voltage, (given in millivolts), peak-power, continuous-use-power (both expressed in milliamperes supplied at the stated voltage), and conversion-efficiency (expressed in percent).

`"power-sources-names"` S

Standard *property name* defining the platform power source names.

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

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

`"platform-battery-sources"` S

Standard *property name* defining the batteries utilized by a platform.

prop-encoded-array: an array of integers, each encoded as with **`encode-int`**.

Each value in this array is the manufacturer's rated capacity of the battery expressed in milliwatt-hours.

`"battery-sources-names"` S

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

prop-encoded-array: an array of strings, each encoded as with **`encode-string`**.

Each entry in this array corresponds one-for-one with the batteries defined in the **`"platform-battery-sources"`** property.

B.8.1.2 Properties of the power-management-events node

`"power-type"` S

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

prop-encoded-array: an array of integers, each encoded as with **`encode-int`**.

B.8.2 Device Properties

`"power-domains"` S

Standard *property name*, indicating the power domains of which this device is a member.

prop-encoded-array: List of one or more *domain-id's* to which this device belongs. *Domain-id's* is encoded as with **`encode-int`**.

The **"power-domains"** property should only list the *domain-id's* of the lowest power domain tree nodes in which this device has membership. If the device is a member of the default power domain 0 alone, this property does not need to be provided.

"device-power-states" S

Standard *property name* which describes the power states this device supports.

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

This property shall be provided for each physical device which has multiple power states, if platform firmware provides device power state information.

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

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

Table 250. Semantics of device state values

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

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

The interaction of the defined semantics of device power state and domain power level is defined in Table 251. Combinations of Device Power State/Domain Power Level. Those combinations not marked are disallowed.

Table 251. Combinations of Device Power State/Domain Power Level

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

"device-state-transitions" S

Standard *property name* that describes the legal power state transitions supported by the device.

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

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

"power-sources" S

Standard *property name* which designates this device as a consumer of power sourced from a defined power source.

prop-encoded-array: an array of integers, each encoded as with **encode-int** that gives the list of power sources to which this device is connected.

The values are indices into the platform-power-sources data structure. This property shall be provided if platform firmware provides device power state information.

"power-management-mapping" S

Standard *property name* that defines device power states and commands.

prop-encoded-array: an array of integers as encoded with **encode-int**.

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

B.8.2.1 Properties for Power Domain Control Points

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

"controls-power-domains" S

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

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

A single device may serve as multiple logical control points.

B.8.3 Power Management Related Methods

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

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

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

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

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

`system-off` (--) M

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

B.9 Configuration of Platform Resources

Any computer platform is composed of standard components which are invariant (platform 'built-in' standard I/O and power management), optional components which are detectable (a second processor, for example), and configurable components which are self-identifying (system memory, for example). Most computer platforms also provide one or more industry standard I/O buses which allow the insertion of specialized functional adapter cards. These buses generally support a method for automatic identification, interrogation, and option selection of installed adapter cards.

A Platform shall also have the capability of configuring power management resources, if power management is implemented by the platform, as defined in Section B.9.1, "Power Management Resource Configuration," on page 734

B.9.1 Power Management Resource Configuration

For a platform which supports device power management, all platform power management related information shall be resident in the OF device tree prior to the transfer phase of software operation (see the definition of transfer phase in Chapter 2, "System Requirements," on page 41). Dummy devices shall be placed in the device tree for all standard I/O bus connectors which are not in use to provide a node to assign the slot-names, power-domains, and power-sources properties.

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

B.9.1.1 Power Management Information Utility

Any platform capable of being expanded via the addition of power-managed devices shall provide a device power management information utility. The purpose of the utility is to allow a person (end-user or system developer) to enter

power management related device properties of plug-in adapters and peripherals which have no mechanism to automatically report this information to firmware or system software. The need for this utility will disappear as standard protocols are developed for interrogating pluggable adapters and devices to provide power management related information.

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

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

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

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

B.9.1.2 PM Configuration Process

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

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

B.9.1.3 PM Configuration Format

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

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

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

The following table specifies the format of the header:

Table 252. Power Management Configuration Data Header

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

The PM Partition data area format value shall be 1.

The following table specifies the format of the data blocks:

Table 253. Data Block Format

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

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

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

Table 254. Node Data Block Format

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

Table 255. Property Data Block Format

Field	Size	Description
Block_type	1 byte	This field shall contain the value 0x02
Node_path	2 bytes	Pointer to a NULL terminated string giving the path name of the node to which the designated property list belongs.

Table 255. Property Data Block Format *(Continued)*

Field	Size	Description
Property_list_ptr	2 bytes	Pointer to a NULL terminated string containing the property list to be assigned to the designated node.
Reserved	3 byte	Reserved

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

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

B.10 Client Program Requirements

For LoPAPR platforms, the client program requirements are defined in Appendix C, "PA Processor Binding," on page 753, with the following modifications. OF Client Programs for an LoPAPR platform shall execute in 32-bit mode with an OF cell size of 1.

B.10.1 Load Address

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

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

B.10.2 Initial Register Values

The "Initial Register Values" specified in the PA Binding (see Appendix C, "PA Processor Binding," on page 753) are modified as follows:

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

B.10.3 I/O Devices State

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

All Devices - no DMA and not interrupting

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

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

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

B.10.4 Client Program Format

The data format of a client program compliant with this specification shall be either ELF (Executable and Linkage Format) as defined by *System V Application Binary Interface, PowerPC Processor Supplement* [15], and extended by Section B.10.4.1.1, “ELF Note Section,” on page 738, or PE (Portable Executable) as defined by *Peering Inside the PE: A Tour of the Win32 Portable Executable File Format* [13]. The standard ELF format contains explicit indication as to the program's execution modes (e.g., 32- or 64-bit, Big- or Little-Endian). LoPAPR only supports the 32-bit version (i.e., ELFCLASS32) for 32 and 64 bit platforms.

Note: Other client program formats may be supported, in an implementation specific manner, by an OF 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 PA 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.

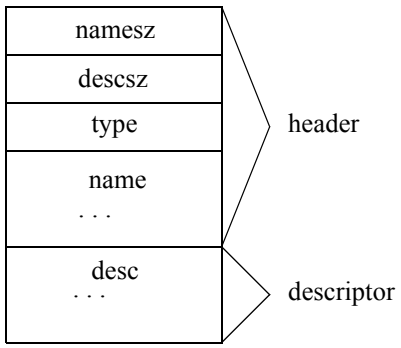
B.10.4.1 ELF-Format

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

B.10.4.1.1 ELF Note Section

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

As defined by *System V Application Binary Interface, PowerPC Processor Supplement* [15], 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 OF struct as:

```
struct          \ Note Section header for OF
/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
```

B.10.4.1.1.1 1275 PowerPC Note Definition

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

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

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

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

Note: DATA field of the file.

B.10.4.1.1.2 1275 IBM,RPA-Client-Config Note Definition

The ns.name field of the LoPAPR Client Program Configuration Note Section shall be `"IBM,RPA-Client-Config"`; the ns.type field shall be 0x12759999.

The format and requirements associated with this ELF Note Section are designed to allow for expandability of the section definition (by adding fields to the end of the section) while retaining forward and backward compatibility for both the 1275 firmware and Client Program. When the 1275 firmware code recognizes the `"IBM,RPA-Client-Config"` note, it creates a property named **`"ibm,rpa-client-config"`** within the **`/chosen`** node reads into this property and interprets the lesser of the descriptor size or the maximum size of the descriptor that was defined when the firmware was built. Should the note contain a smaller descriptor than was defined when the firmware was built, the firmware assumes default values for the missing descriptor fields. In this way, new fields may be defined, and the four cases of firmware/client program work as follows:

New Firmware/New Client Program:

Client Program Header Note contains old plus new fields.

Firmware reads all the new header and places it in **`"ibm,rpa-client-config"`** property.

Client Program gets feed back that new fields were interpreted by reading property in **`/chosen`**.

Old Firmware/Old Client Program:

Client Program Header Note contains old fields.

Firmware reads all the old definition header and places it in **`"ibm,rpa-client-config"`** property.

Client Program gets feed back that the expected fields were interpreted by reading property in **/chosen**.

New Firmware/Old Client Program:

Client Program Header Note contains only old fields.

Firmware reads only the descriptor length defined in the note header, and places it in **"ibm,rpa-client-config"** property.

Client Program gets feed back on the fields that were interpreted by reading property in **/chosen**.

Firmware uses default values for any missing fields.

Old Firmware/New Client Program:

Client Program Header Note contains old plus new fields.

Firmware reads only the length that it was defined when it was built, and places it in **"ibm,rpa-client-config"** property.

Client Program gets feed back that new fields were interpreted by reading property in **/chosen**, those missing fields indicate function not implemented by the platform.

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 the ELF header note named IBM,RPA-Client-Config of type 1275, the format of the Note Section's descriptor can be described by an OF struct, as follows:

```
struct          \ Note Section descriptor for OF
/L      field       ns.lparaffinity         \= "0/1" (default assumption to be "N")
/L      field       ns.min-rmo-size         \Minimum size of the Real Mode Accessible Storage Area in MB
/L      field       ns.min-rmo-percent      \Minimum percentage of total storage that must be Real Mode Accessible
/L      field       ns.max-pft-size         \Maximum size of the hardware page frame table as a power of 2
/L      field       ns.splpar               \= "0/1" (default assumption to be "N")
/L      field       ns.min-load             \The minimum amount of code that must be loaded at load-base.
                                            \ (default value -1)
/L      field       ns.new-mem-def          \Flag to indicate use of ibm,dynamic-reconfiguration-memory definition.
                                            \ (default value 0)
/L      field       ns.ignore-my-client-config      \Flag: 1 = do not change boot configuration variables based
                                                    \upon the values in this header.
                                                    \(default value 0)
/L      field       ns.large-page-ready     \Flag to indicate the partition OS is prepared for large pages.
/L      field       ns.force_alpha_mode
```

Note: The size of the /L field is based off of e_ident (EI_CLASS) i.e. is 4 for ELFCLASS32.

The **ns.lparaffinity** field is a binary flag whose valid values are 0 or 1. If the field is not one of these valid values the value is assumed to be 0. If the character value is 1, the client program requests that the platform provide all available affinity information.

The **ns.min-rmo** field specifies the minimum amount of real mode addressable storage (in bytes times 2^{20}) that the client program needs to operate. The **ns.min-rmo-percent** field specifies the minimum percentage (valid values 0-100) of storage that must be real mode addressable for the client program to operate. The platform shall start the client program with a quantity of real mode accessible storage (starting at location 0) of at least the ceiling of these two values.

The **ns.max-pft-size** field value specifies the largest hardware Page Frame Table (in bytes times $2^{\texttt{ns.max-pft-size}}$) that the client program can support. The firmware shall not start a client program with a PFT larger than this amount The minimum value is 18, the platform ignores the field if the value is less than 18 and uses the platform defined default value.

The **ns.splpar** field is a binary flag whose valid values are 0 or 1. If the field is not one of these valid values the value is assumed to be 0. If the field's value is 1, the client program supports running in shared processor logical partitioning mode. If the character value is not 1 and the partition is running in shared processor mode, platform firmware reports a platform-specific error code and halts the boot. However, if the client-program does not contain an IBM,RPA-Client-Config note, firmware assumes the OS supports shared processor logical partition mode. This exception only applies to the **ns.slpar** field.

The **ns.min-load** field specifies the minimum amount of the client program load module that must be loaded at **load-base**. If this value is a -1 then the entire load module must be loaded starting at **load-base** else the client program load fails. The default value is assumed to be -1. If the value of is greater than the platform can support client program load fails. Given that the platform can load the minimum amount of the client program load module at **load-base**, it loads the amount up to the boundary specified by **ns.min-load** starting at **load-base**, then it loads the rest of the load module into contiguous storage at a location selected by platform firmware (default, if possible, is that the residual is loaded immediately following the first segment resulting in a single segment load).

The **ns.new-mem-def** field is a flag which indicates if the ibm,dynamic-reconfiguration-memory representation of reconfigurable memory may be used. The default value 0x00000000 indicates the new definition may not be used. The value 0x00000001 indicates the new definition may be used. All other values are reserved for future use.

The **ns.large-page-ready** field is a flag which indicates if the partition OS is prepared to support large pages. The default value 0x00000000 indicates that the OS is not prepared for large pages. The value 0x00000001 indicates that the OS is prepared for large pages. All other values are reserved for future use.

If this variable indicates that the OS is not prepared for large pages and large pages are present in the partition configuration, platform firmware reports a platform-specific error code which indicates this mismatch between the partition configuration and the OS capabilities, removes all large pages from the device tree, and continues the OS boot.

If the value of the **ns.ignore-my-client-config** variable is 0x00000001, platform firmware must not examine the value of **ns.large-page-ready** until the client program calls the PROCESS-ELF-HEADER method. The decision to continue boot should then be made based on the value of the **ns.large-page-ready** flag in the updated ELF head passed by this method.

The **ns.force_alpha_mode** field is a flag which indicates that a non-HMC managed I/O services partition with partition management support (VMC) configuration is being requested. The default value of 0x00000000 indicates that the client expects to run in a configuration which is not an I/O services partition configuration. If the partition configuration is not compatible with this setting, the system will be rebooted as a single partition which owns all of the system resources. On reboot, the original partition configuration will be reinstated. The value 0x0000001 indicates that the client is expecting to be executed in a non-HMC managed I/O services partition with partition management support (VMC). If the partition is not in this mode, the system will be rebooted in this mode. In the case that the ns.force_alpha_mode flag is compatible with the partition configuration, the boot process will continue. This flag will be ignored when the system is HMC managed.

B.10.4.1.2 Recognizing ELF-Format Programs

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

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

- a) The "e_ident" field (at offset 0) contains the string "\7fELF", where '\7f' is a byte whose value is (hex) 7f. This indicates the beginning of an ELF file header.
- b) The "EI_CLASS" field (at offset 4) contains the value 1. This indicates the 32-bit variant of the ELF format.
- c) The "e-type" field (at offset 16) contains the value 2. This indicates that the ELF image is executable.

- d) The “e_machine” field (at offset 18) contains the value 20. This indicates that the ELF image is for the PA instruction set.
- e) The “e_version” field (at offset 20) contains the value 1.
- f) The “e_flags” field (at offset 36) contains the value 0.

B.10.4.1.3 Preparing ELF-Format Programs for Execution

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

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

- a) Search for an ELF “note” section of type “1275” as defined in the section “ELF Note Section”. If one is found, and the values specified by its descriptor do not match the firmware's current operating mode, set the appropriate configuration variables to the values specified in the note section descriptor, and restart the firmware so that it will re-execute the **`boot`** command that resulted in the execution of **`init-program`**.
- b) Set the p_paddr field for each PT_LOAD segment equal to its p_vaddr field value if **`real-mode?`** is false and p_paddr is -1. This effectively maps these segments v=r.
- c) Allocate and map, if required, sufficient physical memory for all program segments of type PT_LOAD (i.e. whose “p_type” field contains the value 1) listed in the ELF image's program headers. Note that all PT_LOAD program segments that have a p_paddr value that matches their location in physical memory need not be moved, but the memory that they occupy must be claimed. This special case is added to allow large program images to be loaded without the 2x memory required to move the segments.
- d) Copy the program headers to a “safe” location to guard against the possibility of them being overwritten by the following steps.
- e) For each program segment of type “PT_LOAD”:
  - 1) Copy, if required, the initialized portion of the program segment from its current location in the loaded image to the location given by the section's “p_paddr” field.
  - 2) Fill the rest of the segment with zero bytes (i.e., fill “p_memsz - p_filez” bytes beginning at the address “p_paddr + p_filesz”).
  - 3) If **`real-mode?`** is false, then map the program segment to the virtual address specified by p_vaddr.
- f) Set the saved program state so that subsequent execution of **`"go"`** will begin execution at the address given by the “e_entry” field in the ELF file header. The e_entry field is a physical address if `real-mode?` is **`true`** and is a virtual address if **`real-mode?`** is **`false`**.

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

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

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

B.10.5 Additional Client Interface Requirements

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

B.10.5.1 Client Interface Callbacks

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

B.10.5.1.1 Real-Mode Memory Management Assist Callbacks

claim_mem

IN: [address] min_addr, [address] max_addr, size, align

OUT: throw-code, error, [address] real_addr

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

release_mem

IN: [address] phys, size

OUT: throw-code

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

B.10.5.1.2 Virtual Address Translation Assist Callbacks

alloc_virt_mem

IN: size

OUT: throw-code, error, [address] virt_addr

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

free_virt_mem

IN: [address] virt_addr, size

OUT: throw-code

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

claim_virt

IN: size, align

OUT: throw-code, error, [address] virt_addr

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

release_virt

IN: [address] virt, size

OUT: throw-code

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

B.10.5.2 Client Interface Services

OF shall provide the following *Client Interface Service*:

test-method

IN: phandle, [string] method

OUT: missing-flag?

Tests whether the package method named *method* exists in the package *phandle*. *missing-flag?* is FALSE (0) if the method exists or TRUE (-1) if the method does not exist.

OF may provide the following Client Interface Service:

ibm,enable-ci64

IN: none

OUT: none

After the successful invocation of this method, all Client Interface calls will utilize 64 bit cell items in their argument arrays. This does not affect how the device tree is presented, which will still assume that a cell is 32 bit in the property values. The method returns using the cell size in which it was called. This method exists only on platforms that present the **"ibm,enable-ci64-capable"** property in the **root** node.

B.11 Support Packages

This section describes the LoPAPR Binding specific requirements of OF support packages. These support packages are **disk-label** and **tape-label**. For "network" and/or **obp-tftp** extensions, refer to *Open Firmware: Recommended Practice - TFTP Booting Extensions, Version 0.8* [10]. These packages support the loading and executing of a client program. Another means of executing a Client Program is provided when an OS ROM is a "bootable device" (Refer to Section B.6.2.4, "ROM Node(s)," on page 688, as an example).

B.11.1 “disk-label” Support Package

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

The name of the partition (and, a file contained within the partition) can be explicitly specified by the user via the **`load`** or **`boot`** command, or can be implicitly specified by the value of the **`“boot-device”`** property of the **`/options`** node. The partition and filename are the ARGUMENTS portion of the final COMPONENT of the PATH_NAME, as described in section 4.3.1 of *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2].

The syntax for explicit partition/filename specification is given in section Section B.11.1.2, “Open Method Algorithm,” on page 746 below where partition identifies the partition to be used and filename is the name of a file within that partition. If partition is omitted, the default partition (as determined by the partition format) is used. If filename is omitted, the default filename (i.e., the filename component of the **`boot-device`** path-name) is used.

B.11.1.1 Media Layout Format

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

- FAT (FAT12 and FAT16 File System)
- FDISK (Partitions 4, 5, 6, 0x41 and 0x96)
- ISO-9660 (9660 File System)
- UDF

An LoPAPR platform may choose to support the following media layout formats for historic reasons:

- Mac OS (MAC Binary Image)

B.11.1.1.1 FDISK Partition Types

The following FDISK partition types shall be supported:

Partition Type 4: FAT 12 or FAT 16 File System

Partition Type 5: Extended Chained Partitions

Partition Type 6: Extended Partitions

Partition Type 0x41: Single program image

Partition Type 0x96: ISO 9660 File System

Partition Type 0x??: UDF File System

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

B.11.1.2 Open Method Algorithm

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

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

[partition][,[filename]]

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

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

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

Note: For historic reasons, the following algorithm includes support for the optional MAC OS media layout format.

```
1   OPEN algorithm
2       Set D.SIZE to -1
2       If ARGUMENT$ is not the NULL string and the first character of ARGUMENT$ is in the range '0'--'9' or is
        equal to ','
3           LEFT-PARSE (ARGUMENT$) -> PARTITION$, FILENAME$
2       Else
3           Set PARTITION$ to the NULL string
3           Set FILENAME$ to ARGUMENT$
2       If PARTITION$ is not the NULL string
3           If PARTITION$ is not a decimal number
4               Return FALSE
3           DECIMAL_STRING_TO_NUMBER (PARTITION$) -> PARTITION
3           If PARTITION is 0
4               GET_DISK_SIZE
3           Else
4               Read the first 512 bytes of the device into a buffer
4               SELECT_EXPLICIT_PARTITION (PARTITION)
4               If SELECT_EXPLICIT_PARTITION returned an error indication
5                   Return FALSE
2       Else \ PARTITION$ is NULL
3           Read the first 512 bytes of the device into a buffer
3           SELECT_ACTIVE_PARTITION
3           If SELECT_ACTIVE_PARTITION returned an error indication
4               Return FALSE
2       \ (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.)
2       Call parent’s “seek” method with an argument of 0,0.
2       Return TRUE
```

CHECK_FOR_BPB procedure

If the first four(4) bytes are EBCDIC 'IBMA'(hex character string C9C2D4C1), then the sector does not contain a BPB.

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

CHECK_FOR_ISO_9660 procedure

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

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

CHECK_FOR_FDISK procedure

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

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

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

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

CHECK_FOR_MAC_DISK procedure

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

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

INTERPOSE_BY_TYPE procedure

```
If FILENAME$ is not the NULL string
    If PARTITION-TYPE is 0x96
        INTERPOSE[11](ISO-9660 File System package, FILENAME$)
    Else
        If PARTITION-TYPE is FAT,
            INTERPOSE (FAT File System package, FILENAME$)
```

SELECT_ACTIVE_PARTITION (PARTITION) procedure

```
CHECK_FOR_BPB
    If the buffer contains a BPB
        Set OFFSET to 0
        Set D.SIZE to the maximum size of the disk in bytes, as indicated by the information in the BIOS Parameter Block
        If FILENAME$ is not the NULL string
            INTERPOSE (FAT File System package, FILENAME$)
        Return OKAY
CHECK_FOR_FDISK
If the buffer contains an FDISK partition map
    Search the FDisk partition map, reading new 512-byte sectors into the buffer if necessary to “chain” to extended partition entries (i.e. ones whose type byte at offset 4 contains “5”) for the first (i.e., at the lowest offset) partition entry whose “bootable” field (at offset 0 in the partition entry) contains 0x80.
    If a “bootable” partition was found:
        Set PARTITION-TYPE to that entry's “type” field (the byte at offset 4)
```

```
4                   Set D.OFFSET to the byte offset from the beginning of the disk of the beginning of the partition
                    denoted by that entry.
4                   Set D.SIZE to the size of the partition denoted by that entry.
4                   INTERPOSE_BY_TYPE
4                   Return OKAY

\ (If this point is reached, no partition was marked “bootable”)
3               Search the FDisk partition map beginning in 512-byte sector 0,reading new 512-byte sectors into the buffer
                if necessary to “chain” to extended partition entries, for the first partition (i.e., at the lowest offset) entry
                whose “type” byte is neither 0 nor 5 (5 is the type code that indicates a “chained” extended partition entry).
3               If one is found:
4                   Set PARTITION-TYPE to that entry's “type” field (the byte at offset 4)
4                   Set D.OFFSET to the byte offset from the beginning of the disk of the beginning of the partition
                    denoted by that entry.
4                   Set D.SIZE to the size of the partition in bytes denoted by that entry.
4                   INTERPOSE_BY_TYPE
4                   Return OKAY
3               Else \ (If this point is reached, the partition map did not contain any valid partition entries)
4                   Return ERROR
2           CHECK_FOR_ISO_9660
2           If it is an ISO 9660 disk
3               GET_DISK_SIZE
3               If FILENAME$ is not the NULL string
4                   INTERPOSE (ISO-9660 File System package, FILENAME$)
3               Return OKAY
2           CHECK_FOR_MAC_DISK
2           If this is a Mac partitioned disk
3               Search the Mac partition table for the first “bootable” partition. A partition is “bootable” when the
                pmPartStatus flags indicate that this is a valid, allocated, readable and bootable partition and the
                pmProcessor field contains “powerpc” (using case-insensitive matching).
3               If a Mac “bootable” partition is found
4                   If FILENAME$ is “%BOOT”
5                       If the Nth partition is marked bootable
6                           Set D.OFFSET to the byte offset from the beginning of the disk to the beginning of the
                            boot area, as given by the pmLgBootStart field.
6                           Set D.SIZE to the size of the partition in bytes denoted by pmBootSize.
6                           Return OKAY
4                   Else
5                       If the FILENAME$ is the NULL string
6                           Set D.OFFSET to the byte offset of the “real” partition data
6                           Set D.SIZE to the size of the “real” partition data
5                       Else
7                               INTERPOSE_BY_TYPE
5                       Return OKAY
3               Else
4                   Return ERROR
3               (If this point is reached, no “bootable” partition was found)
3               Return ERROR
1   GET-DISK-SIZE procedure
2           Set OFFSET to 0
```

```
2       If the parent has a “#blocks” method
3           Execute the parent's “#blocks” and “block-size” methods
3           Set D.SIZE to the product of the numbers they returned
2       Else
3           Set D.SIZE to -1
1   SELECT_EXPLICIT_PARTITION procedure
2       CHECK_FOR_BPB
2       If the buffer contains a BPB
3           If PARTITION is 1
4               Set OFFSET to 0
4               Set D.SIZE to the maximum size of the disk in bytes, as indicated by the information in the BIOS
                Parameter Block
4               If FILENAME$ is not the NULL string
5                   INTERPOSE (FAT File System package, FILENAME$)
4               Return OKAY
3           Else \ Have a BPB, but PARTITION <> 1
4               Return ERROR
2       CHECK_FOR_FDISK
2       If an FDisk partition map is found
3           Search the FDisk partition map beginning in 512-byte sector 0, reading new 512-byte sectors into the buffer
            if necessary to “chain” to extended partition entries, for the Nth, where N is the value of PARTITION,
            partition entry whose “type” byte is neither 0 nor 5 (5 is the type code that indicates a “chained” extended
            partition entry).
3           If the Nth partition is found:
4               Set PARTITION-TYPE to that entry's “type” field (the byte at offset 4)
4               Set D.OFFSET to the byte offset from the beginning of the disk to the beginning of the partition
                denoted by that entry.
4               Set D.SIZE to the size of the partition in bytes denoted by that entry.
4               INTERPOSE_BY_TYPE
4               Return OKAY
3           Else \Nth partition does not exist
4               Return ERROR
2       CHECK_FOR_MAC_DISK
2       If this is a Mac partitioned disk
3           Search the Mac partition map for the Nth partition, where N is the value of PARTITION.
3           If the Nth partition is valid, allocated, and readable
4               If FILENAME$ is %BOOT
5                   If the Nth partition is marked bootable
6                       Set D.OFFSET to the byte offset from the beginning of the disk to the beginning of the
                        boot area, as given by the pmLgBootStart field.
6                       Set D.SIZE to the size of the partition in bytes denoted by pmBootSize.
6                       Return OKAY
5                   Else \Nth partition not “bootable”
6                       Return ERROR
4               Else
5                   If FILENAME$ is not the NULL string
6                       INTERPOSE_BY_TYPE
5                   Return OKAY
3           Else \ (If this point is reached, the partition is invalid)
4               Return ERROR
```

```
2       Else \ (If this point is reached, the partition map is not recognized)
3           Return ERROR
```

This algorithm can be used to locate the correct partition and/or file and/or load image from the specified device. The boot device is selected as described in 7.4.3.2 of *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2]. 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.

B.11.2 tape-label Support Package

The **`tape-label`** Support Package shall support tape as a standard byte device with the set of methods specified in *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], Section 3.7.3. Presence of the bootinfo.txt file is optional.

The **`open`** method shall accept an argument string, where brackets denote an optional component:

[file number]

where the first file on the tape media is located at file number 0.

B.11.2.1 Tape Format

The LoPAPR tape format shall consist of files ending with a file mark (FM). The first block of data will be identified as file 0. The bootinfo.txt file, if present, shall be located on the tape as file 0 (the first file). There shall be only one bootinfo.txt file on the tape media. Refer to Figure 36, “Tape Boot Format,” on page 750 for the LoPAPR Tape format.

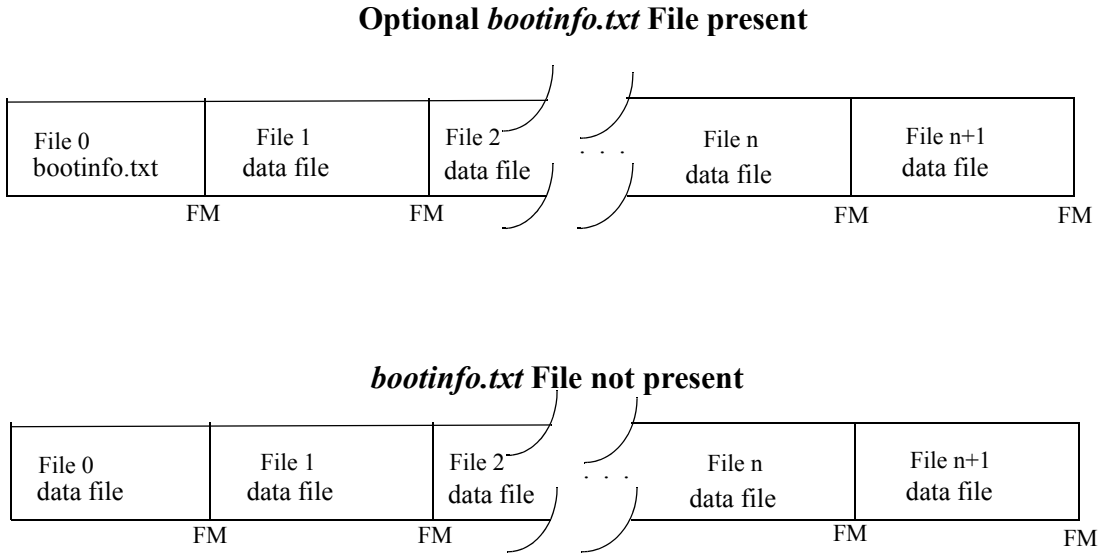


Figure 36. Tape Boot Format

B.11.2.2 Tape *bootinfo.txt* File

The bootinfo.txt file shall have included for each set of **<chrp-boot>** tags a set of **<boot-script>** tags that contains a pointer to the program image to be loaded (Refer to Section B.4.1.6, “Bootinfo Objects,” on page 665). The form for this tape pointer will be:

device specifier = *device*:*file number*

EXAMPLE: device specifier = tape:2 (For the specified set of **<chrp-boot>** tags, load the tape program image from file 2).

A bootinfo.txt file may contain a multiple set of **<chrp-boot>** tags where each one can point to a different tape file number. If a bootinfo.txt file is not present, file 0 should be a bootable file. Only file 0 will be loaded as a bootable image. No other files will be searched if a bootinfo.txt file is not present unless the file number to load is specified by an argument.

B.11.3 network Support Package

The **`network`** Support Package shall adhere to the *Open Firmware: Recommended Practice - TFTP Booting Extensions, Version 0.8* [10] documentation functions and conventions.

B.11.4 Program-image formats.

OF must recognize a client program that is formatted as ELF, as defined in *System V Application Binary Interface, PowerPC Processor Supplement* [15], and PE, as defined in *Peering Inside the PE: A Tour of the Win32 Portable Executable File Format* [13]. Other formats *may* be handled in an implementation-specific manner. *Open Firmware: Recommended Practice - Forth Source and FCode Image Support, Version 1.0* [8] defines the FCode and Forth Program-Image Formats.

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

```
init-program.
set restart? false
if the image is in ELF format (Refer to Section B.10.4.1.2, “Recognizing ELF-Format Programs,” on page 741)
        if the EI_DATA field does not match little-endian?
                set little-endian?  appropriately.
                set restart? true
        locate the PowerPC Note Section
        if the Note Section’s values do not match
                set Configuration Variables appropriately
                set restart? true
        if restart?
                        restart the system, possibly by executing reset-all
        else
                move and/or relocate the ELF image
                (Refer to Section B.10.4.1.3, “Preparing ELF-Format Programs for Execution,” on page 742).
                set GO’s context with initial register values
else if the image is in PE format
        if little-endian? is false
                set little-endian? to true.
                restart the system, possibly by executing reset-all
        else
                move and/or relocate the PE image.
                set GO’s context with initial register values
```

else if the image is FCode Image (hex characters F1,08)
    setup system and do subsequent **`go`** and perform a byte load of the FCode image
else if the image is Forth Code Source Image (ASCII characters \"<space>")
    setup system to **`evaluate`** Forth Source Image
else if the image is a bootinfo.txt file (i.e., begins with "<CHRP-BOOT>")
    setup system to parse the bootinfo.txt file
else
    FAIL, in an implementation-specific manner.

C PA Processor Binding

C.1 Purpose of this Binding

This appendix specifies the application of OF to a PA Processor (which covers all PowerPC processors and their successors), including requirements and practices to support unique firmware specific to a PA Processor. The core requirements and practices specified by OF must be augmented by processor-specific requirements to form a complete specification for the firmware implementation for a PA processor. This appendix establishes such additional requirements pertaining to the processor and the support required by OF.

C.2 Overview

This appendix specifies the application of *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2] to computer systems that use the PA instruction set, including instruction-set-specific requirements and practices for debugging, client program interface and data formats. An implementation of OF for PA shall implement the core requirements as defined in *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2] and the PA-specific extensions described in this binding.

This appendix addresses *Power ISA* [1]. The descriptions that follow, and the relevant sections describing translation features for this binding, assume that the system's PA processor(s) implement the entire PA (that is, all books of *Power ISA* [1]). Some 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.

C.3 Terms

This standard uses the following technical terms.

core, core specification	refers to *IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware, Core Practices and Requirements*
effective address	The 64- or 32-bit address computed by the processor when executing a Storage Access or Branch instruction, or when fetching the next sequential instruction. If address translation is disabled, the real address is the same as the effective address. If address translation is enabled, the real address is determined by, but not necessarily identical to, the effective address.
linkage area	An area within the stack that is reserved for saving certain registers across procedure calls in PA run-time models. This area is reserved by the caller and is allocated above the current stack pointer (`%r1`).
Open Firmware (OF)	The firmware architecture defined by the core specification or, when used as an adjective, a software component compliant with the core specification.
procedure descriptor	a data structure used by some PA run-time models to represent a C "pointer to procedure". The first word of this structure contains the actual address of the procedure.
real address	An address that the processor presents on the processor bus.

Real-Mode — The mode in which all addresses passed between the client and OF are real addresses.

processor bus — The bus that connects the CPU chip to the system.

segmented address translation

The process whereby an Effective Address (EA) is translated into a Virtual Address (VA) and the virtual address is translated into a Real Address (RA). (seeSection C.5.1.2, “Segmented Address Translation,” on page 755 and Book III of *Power ISA* [1] for more detail.)

Table of Contents (TOC)

A data structure used by some PA 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 address (in IEEE 1275 parlance)

The address that a program uses to access a memory location or memory-mapped device register. Depending on the presence or absence of memory mapping hardware in the system, and whether or not that mapping hardware is enabled, a virtual address may or may not be the same as the physical (real) address that appears on an external bus. The IEEE 1275 definition of “virtual address” corresponds to The PA's definition of “effective address.” Except as noted, this document uses the IEEE 1275 definition of virtual address.

Virtual Address (in PA parlance)

An internal address within the PA address translation mechanism, used as in intermediate term in the translation of an effective address to the corresponding real address.

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

C.4 Data Formats and Representations

The cell size shall 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 shall be four-byte aligned (i.e., a multiple of 4).

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

C.5 Memory Management

C.5.1 PA Address Translation Model

This section describes the model that is used for co-existence of OF and client programs (i.e., OSs) with respect to address translation.

The following overview of translation is provided so that the issues relevant to OF for the PA can be discussed. Details that are not relevant to OF issues (e.g., protection) are not described in detail; *Power ISA* [1], particularly Book III, should be consulted for the details. For the scope of this section, terms will be used as defined in *Power ISA* [1].

C.5.1.1 Translation requirements

The default access mode of storage for load and stores (i.e., with translation disabled -- referred to as *Real-Mode*) within the PA 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 OF can run with translation disabled, it must be enabled when performing I/O.

C.5.1.2 Segmented Address Translation

Note: The use of the term Virtual Address in this section refers to the PA definition, while the rest of the document uses the IEEE 1275 definition of virtual address.

Note: The following description of PA address translation is only one of several translation modes available and is given for reference only. See *Power ISA* [1] for complete details.

An Effective Address (EA) of a PA processor is 64(32) bits wide. Each EA is translated into an 80(52)-bit Virtual Address (VA) by prepending a 52(24)-bit Virtual Segment Id (VSID) to the 28 LSbs of the effective address. On 32-bit implementations, the VSID is obtained by indexing into a set of 16 Segment Registers (SRs) using the 4 MSbs of the EA. On 64-bit implementations, the VSID is looked up in a Segment Table using the 36 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 0-67(39) of the VA) into a Real Page Number (RPN) and concatenating this RPN with the byte offset (12 LSbs of the VA). The mapping of VPN to RPN involves a hashing algorithm within a 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.

This process is not performed for every translation! Processors will typically have a Translation Look-aside Buffer (TLB) that caches the most recent translations, thus exploiting the natural spatial locality of programs to reduce the overhead of address translation. 64-bit implementations may also implement a Segment Lookaside Buffer (SLB) for the same reasons. On most PA 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 OF, as discussed below.

C.5.1.3 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 PA. The Block Address Translation involves the use of BAT entries that contain a Block Effective Page Index (BEPI), a Block Length (BL) specifier and a Block Real Page Number (BRPN); the architecture defines 4 BAT entries for data (DBAT entries) and 4 BAT entries for instruction (IBAT entries)[1]. BAT areas are restricted to a finite set of allowable lengths, all of which are powers of 2. The smallest BAT area defined is 128 KB (2^{17} bytes). The largest BAT area defined is 256 MB (2^{28} bytes). The starting address of a BAT area in both EA space and RA space must be a multiple of the area's length.

Block Address Translation is done my matching a number of upper bits of the EA (specified by the BL value) against each of the BAT entries. If a match is found, the corresponding BRPN bits replace the matched bits in the EA to produce the RA.

1.The 601 has a single set of BAT entries that are shared by both instruction and data accesses.

Block Address Translation takes precedence over Segmented Address Translation; i.e., if a mapping for a storage location is present in both a BAT entry and a Page Table Entry or HTAB, the Block Address Translation will be used.

Note: Block Address Translation is a deprecated translation mode of the PA. This description is retained here for historical reference. See *Power ISA* [1] for details on all supported addressing mechanisms.

C.5.2 OF's use of memory

OF shall use the memory resources within the space indicated by the **`real-base, real-size, virt-base`** and **`virt-size`** Configuration Variables defined for the PA. As described in the applicable platform binding, a mechanism is defined to enable OF to determine if its current configuration is consistent with the requirements of the client.

If the client program has specific requirements for physical memory or address space usage, it may establish requirements for OF's physical and/or virtual address space usage by means of its program header. When OF loads the client program, it inspects the program header, and if its current usage of physical memory or virtual address space conflicts with that specified in the program header, OF shall set the **`real-base`**, **`real-size`**, **`virt-base`**, and **`virt-size`** to the configuration variables as specified in the header and restart itself. **`Real-base`**, **`real-size`**, **`virt-base`**, and **`virt-size`** may be specified as -1, in which case the firmware is permitted to choose appropriate values for the variables specified as -1.

If the values of the **`real-size`** and/or **`virt-size`** configuration variables do not provide sufficient memory and/or virtual address space for the firmware's own use, then the firmware shall not attempt to load a client program and the condition should be reported to the user. The possibility of not being able to comply with limitations on firmware's size should be tested as the firmware is coming up in order to handle the possibility that a user established an unworkable limitation on the size. Clients can minimize this exposure by setting size to -1 and allowing OF to choose the size.

A PA OF binding shall support two different addressing models, depending upon the setting of the **`real-mode?`** Configuration Variable. This variable indicates the OF 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?`**. OF determines its addressing mode using the value of **`real-mode?`**. If the current state of **`real-mode?`** (and hence, the current state of OF) is incorrect, it shall set **`real-mode?`** appropriately and reset itself, possibly by executing **`reset-all`**.

Memory that cannot be allocated for general purpose use, for example physical memory on LoPAPR systems used for interrupt vectors and implementation specific areas, shall not appear in the "**`available`**" property of the **`memory`** node. A Client Program that needs to use such memory for its architected purpose must not claim that area prior to use.

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

C.5.2.1 Real-Mode

In Real-Mode (when **`real-mode?`** is **`true`**), the use of address translations by OF and its client are independent. Either they do not use translation, or their translations are private; they do not share any translations. All interfaces between the two must pass the real address of the data. Any data structure shared by OF and its client that refers to *virt* addresses in *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], or this binding, must be real addresses.

Note: In particular, that the address of the Client interface handler, that is passed to the client, has to be a real address.

The Configuration Variables **real-base** and **real-size** should indicate the physical memory base and size in which OF must locate itself. In Real-Mode, the Configuration Variables **virt-base** and **virt-size** do not have meaning and should be set to -1.

C.5.2.2 Virtual-Mode

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

The Configuration Variables **virt-base** and **virt-size** should indicate the virtual address space base address and size that OF should use. The Configuration Variables **real-base** and **real-size** should indicate the physical memory base and size in which OF must locate itself.

C.5.2.3 Device Interface (Real-Mode)

While OF is performing system initialization and probing functions, it establishes and maintains its own translations. In particular, it maintains its own Page Tables (and/or BAT entries) 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.

C.5.2.4 Device Interface (Virtual-Mode)

OF will establish its own translation environment, handling DSI/ISI interrupts as in the Real-Mode case. However, this environment will, in general, contain translations in which virtual addresses do not equal real addresses. The virtual address space used by OF 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 client OS decides that it no longer requires the services of OF.

C.5.2.5 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 shall be disabled before the client interface call is made.

OF 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), OF shall save the state of all relevant translation resources (e.g., SDR1, BATs) and restore them before returning to the client. Likewise, it *may* take over interrupts for its own use (e.g., for doing “lazy” allocation of BATs); it shall 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 shall ensure that interrupts are disabled before calling the Client Interface handler and call the handler from only one CPU at a time. The client shall also ensure that other processors do not generate translation exceptions for the duration of the call.

Client programs are not required to assume responsibility for physical memory management. The client program must use the OF claim client interface service to allocate physical memory while physical memory is managed by OF. Physical memory shall remain managed by OF until the client program defines the real-mode physical memory management assist callbacks. Physical memory must be managed by the client program once the client program defines the real-mode physical memory management assist callbacks. OF shall use the client program's real-mode physical mem-

ory management assist callbacks to allocate physical memory after the client program has assumed physical memory management.

In Real-Mode, **claim** methods shall not allocate more pages than are necessary to satisfy the request.

C.5.2.6 Client Interface (Virtual-Mode)

Client interface calls are essentially "subroutine" calls to OF. 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 OF internal 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 OF 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., Page Tables), possibly handling spill conditions, OF cannot know how to perform these updates.

Hence, there shall be **callback** services provided by the client for use by OF for such actions; see Section C.9.5.1, "Real-Mode physical memory management assist callback," on page 780.

In order to let clients (i.e., target OSs) know where OF lives in the address space, the following rules shall be followed by an OF implementation for the PA and by client programs.

OF:

- OF shall maintain its "translations" "mmu"-node property (see Section C.6.1.7, "Memory Management Unit properties," on page 774)
- OF's **claim** methods shall not allocate more pages than are necessary to satisfy the request.
- When a client executes **set-callback**, OF shall attempt to invoke the "translate" callback. If the translate callback is implemented, OF shall cease use of address translation hardware, instead using the client callbacks for changes to address translation.

The **exit** service must continue to work after a **set-callback** that takes over address translation. This implies that OF takes responsibility for address translation hardware upon **exit** and must maintain internal information about translations that it requests of the client.

Client Programs:

- Client programs that take control of the management of address translation hardware and expect to be able to subsequently invoke OF client services must provide callbacks to assist OF in address translation (see Section C.9.5.1, "Real-Mode physical memory management assist callback," on page 780).
- A client program shall not directly manipulate any address translation hardware before it either a) ceases to invoke OF client services or b) issues a **set-callback** to install the "translate" callback.

Note: The intended sequence is that a client program will first issue a set-callback and then take control of address translation hardware. Address translation hardware includes BAT entries, page table, segment registers, Machine State Register and the interrupt vectors relating to translation faults.

C.5.2.7 User Interface (Real-Mode)

In Real-Mode, OF regains total control of the system. As with the Client interface in Real-Mode, it should save the state of the translation resources (including interrupt vectors) upon entry and should restore them upon exit.

C.5.2.8 User Interface (Virtual-Mode)

When the User interface is invoked, OF 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, this binding defines **`callback`** services that the client provides for use by OF; see Section C.9.5.1, “Real-Mode physical memory management assist callback,” on page 780.

C.6 Properties

This section describes the standard properties of a PA OF implementation.

C.6.1 CPU properties

C.6.1.1 The Device Tree

OF requires that the multiple instances of any device that appears more than once in the device tree must be distinguishable by means of their **`"reg"`** properties. The **`"reg"`** property must express the address of each node relative to its parent bus. Furthermore, the core specification says that the root node of the device tree usually represents the main physical bus of the system. Thus, if processors are not directly addressable on the main physical bus, as is expected to be the case on many/most PA-based systems, the CPU nodes on such systems may not be children of the root node but must instead be children of a pseudo-device node. In this case, the name of the pseudo-device node, which will usually be a child of the root node, shall be “cpus”.

The “cpus” node shall have one child node of device_type “cpu” for each processor.

C.6.1.2 Physical Address Formats and Representations for CPU Nodes

C.6.1.2.1 Numerical Representation

The numerical representation of a processor’s “address” in a LoPAPR system shall consist of one cell, encoded as follows (Bit# 0 refers to the least significant bit)**:**

Table 256. Numerical Representation of a Processor's "address"

Bit#	33222222 10987654	22221111 32109876	11111100 54321098	00000000 76543210
phys.lo cell:	00000000	00000000	00000000	pppppppp

where: *pppppppp* is an 8-bit integer representing the interprocessor interrupt identifier used by the platform.

C.6.1.2.2 Text Representation

The text representation of a processor's "address" shall be an ASCII hexadecimal number in the range **`0...FF`**.

Conversion of the hexadecimal number from text representation to numeric representation shall be case insensitive, and leading zeros shall be permitted but not required.

Conversion from numeric representation to text representation shall use the lower case forms of the hexadecimal digits in the range **`a..f`**, suppressing leading zeros.

C.6.1.2.3 Unit Address Representation

A processor's "unit-number" (i.e. the first component of its **`"reg"`** value) is the interprocessor interrupt destination identifier used by the platform. For a uni-processor platform, the "unit-number" shall be zero.

C.6.1.3 CPUS Node Properties

The following properties shall be created within the "cpus" node.

`"#address-cells"`

Standard *property name* to define the number of cells required to represent the physical addresses for the "**`cpu`**" nodes (i.e., the children of the "**`cpus`**" node).

prop-encoded-array: Integer constant 1, encoded as with **`encode-int`**.

The value of **`"#address-cells"`** for the "cpus" node shall be 1.

`"#size-cells"`

Standard *property name* to define the number of cells necessary to represent the length of a physical address range.

prop-encoded-array: Integer constant 0, encoded as with **`encode-int`**.

The value of "**`#size-cells`**" for the "cpus" pseudo-device node is 0 because the processors that are represented by the cpu nodes do not consume any physical address space.

C.6.1.4 CPU Node Properties

For each CPU in the system, a cpu-node shall be defined as a child of **`"cpus".`** The following properties apply to each of these nodes. The **`cpus`** node shall not have "**`reg`**" or "**`ranges`**" properties. In general, properties in a cpu-node that affect the software interface (for example properties that convey the presence of instructions, presence of registers, or location of resources) to the processor are preserved by the device tree once presented upon boot. For a list of properties that may change before a reboot, see Table 122, "Properties of the Nodes That May Be Reported by ibm,update-properties for a "Scope"," on page 252.

"name"

Standard *property name*. The value of this property shall be of the form: "PowerPC,<name>", where <name> is the name of the processor chip which may be displayed to the user. <name> *shall not* contain underscores.

"device_type"

Standard *property name*. The value of this property for CPU nodes shall be "cpu".

"reg"

Standard *proper name* to define a cpu node's unit-address.

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

For a cpu node, the first and only value of the **"reg"** property shall be the number of the per-processor interrupt line assigned to the processor represented by the node. For a uni-processor platform, the value of the **"reg"** property shall be zero.

"status"

Standard *property name*. The value of the is property shall be one of the following string values:

"okay" for a good processor.

"fail" for a processor that fails during power-on testing.

"fail-offline" for a processor that has been automatically deconfigured because of previous failures.

"disabled" for a processor that has been manually deconfigured.

"cpu-version"

property name: Represents the processor type.

prop-encoded-value: The value, encoded as with **encode-int**, shall be either the value obtained by reading the Processor Version Register of the processor described by this node, or the logical processor version as given in Table 257, "Logical Processor Version Values," on page 761. The first byte of the logical processor version value shall be 0x0F. The values of the "Logical Processor Version" column of Table 257, "Logical Processor Version Values," on page 761 indicate that the processor provides the base support described by that version of the architecture. The presence and value of all optional and implementation dependent features and facilities are described by their corresponding properties.

Table 257. Logical Processor Version Values

Logical Processor Version	Property Value
2.04	0x0F000001
2.05	0x0F000002
2.06	0x0F000003
2.06 plus: URG field in DSCR (Bits 55-57)	0x0F100003
2.07	0x0F000004
2.08	0x0F000005

"clock-frequency"

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

"ibm,extended-clock-frequency"

property name: Property that represents the internal processor speed in hertz of this node. This property allows the encoding of multi-giga-hertz quantities.

prop-encoded-array: Consisting of the low order 32 bits of two cells (freq-hi, freq-lo) each encoded as with **encode-int**, such that their combined value is (the low order 32 bits of freq-hi || the low order 32 bits of freq-lo).

"timebase-frequency"

Standard *property name*, encoded as with **encode-int**, that represents the rate (in hertz) at which the PA Time-Base and Decrementer registers are updated.

Note: The 601 PowerPC processor does not have a timebase frequency, therefore on a 601 PowerPC processor the value reported in this property shall be 1 billion (1×10^9) which represents the logical rate of the real time clock.

"ibm,extended-timebase-frequency"

property name: Property that represents the rate in hertz at which the PA TimeBase and Decrementer registers are updated. This property allows the encoding of multi-giga-hertz quantities.

prop-encoded-array: Consisting of the low order 32 bits of two cells (freq-hi, freq-lo) each encoded as with **encode-int**, such that their combined value is (the low order 32 bits of freq-hi || the low order 32 bits of freq-lo).

Note: The **"ibm,extended-timebase-frequency"** property will be deprecated from the architecture due to the emergence of the **"ibm,nominal-tbf"** property and the lack of a need for a two cell version of the **"timebase-frequency"** property. Implementations should not provide the **"ibm,extended-timebase-frequency"** property.

"ibm,nominal-tbf"

property name: Property, encoded as with **encode-int**, that represents the design nominal timebase frequency (in hertz).

"ibm,tbu40-offset"

property name: that provides the value that, when added (ignoring overflow) to the processor TimeBase, yields a value consistent with other platform partitions that utilize their respective values of the property. If the property is missing, the default value is zero.

prop-encoded-array: An eight byte, big endian, unsigned, binary value.

"64-bit"

prop-encoded-array: <none>

This property, if present, indicates that the PA processor defined by this CPU node is a 64-bit implementation of the PA. The absence of this property indicates that the microprocessor defined by this CPU node is a 32 bit implementation of the PA

"64-bit-virtual-address"

prop-encoded-array: <none>

This property, if present, indicates that the PA processor defined by this CPU node supports the 64-bit virtual address subset of the 80-bit virtual address as defined by the PA. The absence of this property indicates that the PA processor defined by this CPU node supports the full 80-bit virtual address defined by the PA. This property is only valid for 64-bit implementations.

Note: The **"64-bit-virtual-address"** will be deprecated from the architecture. Implementations should not provide this property.

"603-translation"

prop-encoded-array: <none>

This property, if present, indicates that the PA processor defined by this CPU node uses the PowerPC 603 processor defined mechanism to update its Translation Lookaside Buffers (TLBs). The absence of this property indicates that the PA processor defined by this CPU node does not use the PowerPC 603 processor defined mechanism to update its TLBs.

"603-power-management"

prop-encoded-array: <none>

This property, if present, indicates that the PA processor defined by this CPU node implements the PowerPC 603 processor defined power management states. The absence of this property indicates that the PA processor defined by this CPU node does not support the PowerPC 603 processor defined power management states.

"bus-frequency"

Standard *property name*, encoded as with **encode-int**, that represents the speed (in hertz) of this processor's bus.

"ibm,extended-bus-frequency"

property name: Property that represents the rate in hertz of this processor's bus. This property allows the encoding of multi-giga-hertz quantities.

prop-encoded-array: Consisting of the low order 32 bits of two cells (freq-hi, freq-lo) each encoded as with **encode-int**, such that their combined value is (the low order 32 bits of freq-hi || the low order 32 bits of freq-lo).

"32-64-bridge"

prop-encoded-array: <none>

This property, if present, indicates that the PA processor defined by this CPU node implements the "Bridge Facilities and Instructions for 64-bit Implementations" as described in an appendix of Book III of *Power ISA* [1]. The absence of this property indicates that the PA processor defined by this CPU node does not support these facilities and instructions.

"external-control"

prop-encoded-array: <none>

This property, if present, indicates that the PA processor defined by this CPU node implements the External Control Facility as described in the "Optional Facilities and Instructions" appendix of Book II of *Power ISA* [1]. The absence of his property indicates that the PA processor defined by this CPU node does not support the External Control Facility.

"general-purpose"

prop-encoded-array: <none>

This property, if present, indicates that the PA processor defined by this CPU node implements the floating point instructions *fsqrt*, *fsqrts* and *stfiwx*. The absence of this property indicates that the PA processor defined by this CPU node does not support the floating point instructions *fsqrt*, *fsqrts* and *stfiwx*.

"reservation-granule-size"

Standard property, encoded as with **encode-int**, that represents the reservation granule size (i.e., the minimum size of lock variables) supported by this processor, in bytes.

"graphics"

prop-encoded-array: <none>

This property, if present, indicates that the PA processor defined by this CPU node implements floating point instructions *fres, frsqrte,* and *fsel*. The absence of this property indicates that the PA processor defined by this CPU node does not support the floating point instructions *fres, frsqrte,* and *fsel.*

"performance-monitor"

property name: Indicates that the processor described by this node implements a performance monitor.

prop-encoded-array: Consists of a pair of values, each encoded as with **encode-int**. The first value of the pair shall be 0 indicating that the performance monitor functionality is implementation specific. The second value of the pair represents the documentation describing the performance monitor functionality implemented by the processor described by this node. The documentation represented by the second value is specified in Table 258, "Documentation for Implementation Specific Performance Monitors," on page 764.

Table 258. Documentation for Implementation Specific Performance Monitors

Second Value	Documentation
0	*Power 5+ Performance Monitor Programmer's Guide*
1	*Power 7 Performance Monitor Programmer's Guide*

"ibm,vmx"

property name that indicates that the processor supports the POWER VMX architecture.

prop-encoded-array: an integer encoded as with **encode-int**, that represents the level of the VMX architecture supported. The first level supported is the value 1. The value of 1 represents the level of support described by the *A Vector/SIMD Multimedia eXtension to the PowerPC Architecture, Specification Revision 1.2.3, 7/18/97* specification. The value of 2 represents the level of support provided by the VSX option of *Power ISA* [1] level 2.06.

"ibm,segment-page-sizes"

property name: that indicates the segment base page sizes and related encodings supported by the processor.

prop-encoded-array: one or more segment-page-size-descriptor(s).

segment-page-size-descriptor: a segment-page-size-header followed by a pte-lp-descriptor.

segment-page-size-header: Consists of three cells (X,Y,Z) encoded as with **encode-int**. The first cell represents the base page size of the segment (the page size which determines the hash value used to locate the segment's page table entries) as 2^X. The second cell contains the SLB encoding that, ORed with the RS register value for use by a slbmte instruction, selects this segment's base page size. Note, the low order bits of the cell Y are aligned with

the low order bits of RS and the RS's L and LP bits are zero prior to the logical OR operation. The third cell contains the number of pte-lp-encodings in the pte-lp-descriptor.

pte-lp-descriptor: Consists of Z (from the segment-page-size-header) pte-lp-encoding(s), one for each of the page sizes supported for this base segment page size.

pte-lp-encoding: Each pte-lp-encoding consists of two cells (P,Q) encoded as with **`encode-int`**. The first cell represents the page size of the encoding as 2^P (thus implying the number of low order RPN bits that are available to page size encoding). The second cell, left shifted 12 bit positions, is the encoding to be entered into the available low order RPN bits to represent this page size for this segment base page size.

Note: A *segment-page-size-descriptor* applies to a segment only if the size of the segment is greater than or equal to all of the page sizes within the *pte-lp-encoding*(s) contained within the *segment-page-size-descriptor.*

`"ibm,processor-page-sizes"`

property name: Relates the number and sizes of the virtual memory page sizes supported by the processor describe by this node.

prop-encoded-array: One to N cells in ascending value order, each encoded as with **`encode-int`**, each cell represents the size of a supported virtual memory page where the value of the cell is the power of 2 of the cell size. i.e. a 4 K page size is represented by the value 12 (4 K= 2^{12}) etc.

`"ibm,processor-segment-sizes"`

property name: Relates the number and sizes of the virtual memory segment sizes supported by the processor described by this node.

prop-encoded-array: One to N cells in ascending value order of the segment selector (SLBE B field), each encoded as with **`encode-int`**, each positive value cell represents the size of a supported virtual memory segment where the value of the cell is the power of 2 of the segment size. That is, a 256Meg segment size is represented by the value 28 (256Meg = 2^{28}) etc. (negative valued cells represent unsupported encodings).

`"ibm,processor-storage-keys"`

property name indicating the number of virtual storage keys supported by the processor described by this node.

prop-encoded-array: Consists of two cells encoded as with **`encode-int`**. The first cell represents the number of virtual storage keys supported for data accesses while the second cell represents the number of virtual storage keys supported for instruction accesses. The cell value of zero indicates that no storage keys are supported for the access type.

`"ibm,processor-vadd-size"`

property name indicating the number of virtual address bits that are supported by the processor described by this node.

prop-encode-array: An integer, encoded as with **`encode-int,`** that represents the number of supported virtual address bits.

Note: A processor described by this node implements the least significant **`"ibm,processor-vadd-size"`** bits of the architected virtual address.

`"ibm,vrma-page-sizes"`

property-name: Maps the VRMASD field values implemented by the processor described by this node to their page sizes.

prop-encoded-array: Array of one or more *VRMA page-size-descriptor*(s) starting with the value selected by the firmware when booting the partition, followed by the other values supported by the platform.

VRMA page-size-descriptor: A pair of cells encoded as with **encode-int**; The first cell is the log base 2 of the page size. The second cell contains, in its low order bits, the VRMASD field value to achieve that supported size. The high order bits of the second cell are zero.

"ibm,estimate-precision"

property name: Relates PA estimate instruction mnemonics to precisions supported by the processor described by this node.

prop-encoded-array: One or more *instruction-precision descriptor*(s).

instruction-precision descriptor: An *instruction descriptor* followed by a *precision descriptor*. An *instruction-precision* descriptor relates one estimate instruction mnemonic to the precision supported by the processor described by this node for that estimate instruction mnemonic.

instruction descriptor: Consists of one PA instruction mnemonic encoded as with **encode-string**.

precision descriptor: Consists of an integer, encoded as with **encode-int**, specifying the number of bits of precision the processor described by this node supports for an instruction mnemonic.

"ibm,dfp"

property name: Indicates that the processor described by this node supports the Decimal Floating Point (DFP) architecture.

prop-encoded-value: an integer, encoded as with **encode-int**, that represents the level of DFP support of the CPU described by this node. The absolute value of the integer represents the level of the DFP architecture supported. The sign of the integer indicates how the architecture level is supported. A positive integer indicates native support while a negative integer indicates emulation assisted support. The absolute values supported are as follows:

1: Represents the level of support defined by Version 2.05 of the *Power ISA* [1].
2: Represents the level of support defined by Version 2.06 of the *Power ISA* [1].

"ibm,purr"

property name: Indicates that the processor described by this node implements a Processor Utilization of Resources Register (PURR).

prop-encoded-value: an integer, encoded as with **encode-int**, that represents the level of PURR architecture supported. The first level supported is the value 1 and is defined by Section 6.5 "Processor Utilization of Resources Register" of Book III of version 2.02 of the PA.

"ibm,spurr"

property name: Indicates that the processor described by this node implements a Scaled Processor Utilization of Resources Register (SPURR).

prop-encoded-value: an integer, encoded as with **encode-int**, that represents the level of SPURR architecture supported. The value of 1 represents the level of support defined by Version 2.05 of the *Power ISA* [1].

"ibm,pa-features"

property name: Indicates level of support of several extended features of the Processor Architecture.

prop-encoded-array: One or more *attribute-descriptor*(s).

attribute-descriptor: Consists of an *attribute-header* immediately followed by an *attribute-specifier.*

attribute-header: Consists of two bytes. The first byte is an unsigned integer representing a value from 1 to 254. The first byte specifies the number of bytes implemented by the platform of the *attribute-specifier.* The second byte is an unsigned integer specifying the *attribute-specifier-type*.

attribute-specifier: The *attribute-specifier* is defined by the *attribute-specifier-type* of the *attribute-header.* The *attribute-specifier* for the *attribute-specifier-type* value of 0 is defined by Table 259, “attribute-specifier definition for attribute-specifier-type value 0,” on page 767.

Table 259. *attribute-specifier* definition for *attribute-specifier-type* value 0

Byte Number	Bit Number	Attribute Name	Description
0	0	Memory Management Unit (MMU)	The value of 1 indicates MMU support; else not supported.
	1	Floating Point Unit (FPU)	The value of 1 indicates FPU support; else not supported.
	2	Segment Lookaside Buffer (SLB).	The value of 1 indicates SLB support; else not supported.
	3	RUN field	The value of 1 indicates support for the RUN field of the Control Register (CTRL, SPR #152); else not supported.
	4	Reserved	Reserved bits within defined bytes *shall* be zero.
	5	Data Address Breakpoint Register (DABR)	The value of 1 indicates DABR support; else not supported.
	6	No Execute (N) bit in Page Table Entries.	The value of 1 indicates No Execute (N) bit in Page Table Entry support; else not supported.
	7	Write Through Required (W) bit	The value of 1 indicates setting the W bit to 1 (write through always) is supported; else attempting to set the W bit to 1 has no effect
1	0	Memory Coherence Required (M) bit	The value of 1 indicates that setting the M bit to 0 (main storage not always coherent) is supported; else attempting to set the M bit to 0 has no effect.
	1	Data Storage Interrupt Status Register (DSISR) set on an alignment interrupt.	The value of 1 indicates that the DSISR is set on an alignment interrupt as described by version 2.01 of PA; else the DSISR is not set on alignment interrupt as described by version 2.01 of PA.
	2	I=1 (cache inhibited) Large Pages	The value of 1 indicates support for I=1 (cache inhibited) large pages; else not supported.
	3	Round to Integer (from floating point) group of instructions.	The value of 1 indicates support for the *frin, friz, frip,* and *frim* instructions; else these instructions are not supported.
	4	Data Address Breakpoint Register Extension (DABRX)	The value of 1 indicates support for the DABRX architecture as defined by version 2.02 of PA; else not supported.
	5	User Accessible SPRG3	The value of 1 indicates support for accessing SPRG3 in Problem State; else SPRG3 is not accessible in Problem State.
	6	Reading an invalid SLB entry returns zeros.	The value of 1 indicates that reading an invalid SLB entry always returns zeros; else non-zero values may be returned.
	7	Support for “110” value of the Page Protection (PP) bits.	The value of 1 indicates support for “110” value of the Page Protection (PP) bits as described by version 2.04 of PA; else “110” is not supported.as described by 2.04 of PA.

Table 259. *attribute-specifier* definition for *attribute-specifier-type* value 0 ***(Continued)***

Byte Number	Bit Number	Attribute Name	Description
2	0	Virtualized Partition Memory (VPM)	The value of 1 indicates support for Virtualized Partition Memory (VPM) as described by version 2.04 of PA; else not supported.
	1	2.05 Data Stream Support	The value of 1 indicates that data streams as described by version 2.05 of PA are supported; else not supported.
	2	Reserved	Reserved bits within defined bytes *shall* be zero.
	3	Data Address Register (DAR) set on an alignment interrupt.	The value of 1 indicates that the DAR is set on an alignment interrupt as described by version 2.01 of PA; else the DAR is not set on alignment interrupt as described by version 2.01 of PA.
	4	Reserved	Reserved bits within defined bytes *shall* be zero.
	5	Program Priority Register (PPR)	The value of 1 indicates that the PPR is implemented as described by version 2.03 of PA; else the PPR is not implemented as described by version 2.03 of PA.
	6	2.02 Data Stream Support	The value of 1 indicates that data streams as described by version 2.02 of PA are supported; else not supported.
	7	2.06 Data Stream Support	The value of 1 indicates that data streams as described by version 2.06 of PA are supported; else the 2.06 version data streams are not supported.
3	0	LSD in DSCR(Bit 58)	The value of 1indicates that “Load Stream Disable” bit of the Data Stream Control Register is implemented
	1	URG in DSCR (Bits 55::57)	The value of 1 indicates that the “Depth Attainment Urgency” field of the Data Stream Control Register is implemented.
	2-7	Reserved	Reserved bits within defined bytes *shall* be zero.
4	Storage Order Options		Byte bits define the availability of specific options
	0	2.06 Strong Storage Order	The value of 1 indicates that Strong Storage Order as defined by version 2.06 of PA is supported; else not.
	1-7	Reserved for future storage order options	Reserved bits within defined bytes *shall* be zero.
5	0	Little Endian	The value of 1indicates support for Little Endian as described by version 2.03 of PA; else not supported.
	1	Come From Address Register (CFAR)	The value of 1 indicates that the CFAR is implemented as described by version 2.05 of PA; else the CFAR is not implemented as described by version 2.05 of PA.
	2	Elemental Barriers	The value of 1 indicates that elemental barriers are supported; else elemental barriers are not supported.
	3	2.07 load/store quadword	The value of 1 indicates that the load/store quadword category as described by version 2.07 of POWER ISA is supported; else the 2.07 version load/store quadword category is not supported.
	4-7	Reserved	Reserved bits within defined bytes *shall* be zero.
6-7	Data Streaming Specifications		
	0	2.07 Data Streaming Support	The value of 1 indicates that data streams as described by version 2.07 of POWER ISA are supported; else the 2.07 version data streams are not supported.
	1-7	Reserved	Reserved bits within defined bytes *shall* be zero.

Table 259. *attribute-specifier* definition for *attribute-specifier-type* value 0 ***(Continued)***

Byte Number	Bit Number	Attribute Name	Description
8-15	0-7	Reserved Co-Processor Option	Individual non-zero bits indicate available coprocessor types per their architected ACOP bit locations. (the value 0x0000000000000000 indicates that moving to/from the ACOP SPR or the ICSWX instruction should not be attempted)
16-17	Level of Vector Category Support		
	0	2.07 Vector Support	The value of 1 indicates that the vector category as described by version 2.07 of POWER ISA is supported; else the 2.07 version vector category is not supported.
	1-7	Reserved	Reserved bits within defined bytes *shall* be zero.
18-19	Level of Vector Scalar Category Support		
	0	2.07 Vector Scalar Support	The value of 1 indicates that the vector scalar category as described by version 2.07 of POWER ISA is supported; else the 2.07 version vector scalar category is not supported.
	1-7	Reserved	Reserved bits within defined bytes *shall* be zero.
20-21	Level of Vector.XOR Category Support		
	0	2.07 Vector.XOR Support	The value of 1 indicates that the vector.xor category as described by version 2.07 of POWER ISA is supported; else the 2.07 version vector.xor category is not supported.
	1-7	Reserved	Reserved bits within defined bytes *shall* be zero.
22-23	Level of Transactional Memory Category Support		
	0	2.07 Transactional Memory Support	The value of 1 indicates that the Transactional Memory Category as described by version 2.07 of POWER ISA is supported; else the 2.07 version Transactional Memory Category is not supported.
	1-7	Reserved	Reserved bits within defined bytes *shall* be zero.
24-255	0-7	Undefined	Readers *shall* ignore undefined bytes if present.

"ibm,pi-features"

property name: Indicates level of support of processor implementation specific options not described by the Processor Architecture.

prop-encoded-array: One or more *pi-attribute-descriptor*(s).

pi-attribute-descriptor: Consists of a *pi-attribute-header* immediately followed by a *pi-attribute-specifier.*

pi-attribute-header: Consists of two bytes. The first byte is an unsigned integer representing a value from 1 to 254. The first byte specifies the number of bytes implemented by the platform of the *pi-attribute-specifier.* The second byte is an unsigned integer specifying the *pi-attribute-specifier-type*.

pi-attribute-specifier: The *pi-attribute-specifier* is defined by the *pi-attribute-specifier-type* of the *pi-attribute-header.* The *pi-attribute-specifier* for the *pi-attribute-specifier-type* value of 0 is defined by Table 260, "'definition for pi-attribute-specifier-type value 0," on page 770.

Table 260. ‘definition for *pi-attribute-specifier-type* value 0

Byte Number	Bit Number	Attribute Name	Description
0	0	P4 Data Address Register (DAR) setting on alignment interrupt.	The value of 1 indicates that the DAR is set on an alignment interrupt as described by version 2.01 of PA except for the case where the interrupt is caused by an unsupported access to cache inhibited space. In this case, the DAR will be set to the effective address of the first access into the cache inhibited space. The value of 0 indicates that the processor does not adhere to this behavior.
	1	Reserved	Reserved bits within defined bytes *shall* be zero.
	2	Ordered Thread Activation/Deactivation	The value of 1 indicates that the **“ibm,ppc-interrupt-server-#s”** property conveys the order that threads need to be activated and deactivated in to achieve optimal performance; else no need to activate and deactivate threads in order.
	3-7	Reserved	Reserved bits within defined bytes *shall* be zero.
1-255	0-7	Reserved	Reserved bits within defined bytes *shall* be zero.

“ibm,negotiated-pa-features”

property name: Indicates level of support negotiated via the **ibm,client-architecture-support** method (See Appendix B, “LoPAPR Binding,” on page 661) of several extended features of the Processor Architecture.

prop-encoded-array: One or more *negotiated-pa-attribute-descriptor*(s).

negotiated-pa-attribute-descriptor: Consists of a *negotiated-pa-attribute-header* immediately followed by a *negotiated-pa-attribute-specifier.*

negotiated-pa-attribute-header: Consists of two bytes. The first byte is an unsigned integer representing a value from 1 to 254. The first byte specifies the number of bytes implemented by the platform of the *negotiated-pa-attribute-specifier.* The second byte is an unsigned integer specifying the *negotiated-pa-attribute-specifier-type*.

negotiated-pa-attribute-specifier: The *negotiated-pa-attribute-specifier* is defined by the *negotiated-pa-attribute-specifier-type* of the *negotiated-pa-attribute-header.* The *negotiated-pa-attribute-specifier* for the *negotiated-pa-attribute-specifier-type* value of 0 is defined by Table 261, “negotiated-pa-attribute-specifier definition for negotiated-pa-attribute-specifier-type value 0,” on page 770.

Table 261. *negotiated-pa-attribute-specifier* definition for *negotiated-pa-attribute-specifier-type* value 0

Byte Number	Bit Number	Attribute Name	Description
0	0	TC Set	The value of 1 indicates that the TC bit is implemented as described by version 2.05 of PA and set to a value of 1; else the TC bit is not implemented as described by version 2.05 of PA or not set to a value of 1.
	1-7	Reserved	Reserved bits within defined bytes *shall* be zero.
1-255	0-7	Reserved	Reserved bits within defined bytes *shall* be zero.

"ibm,raw-pi-features"

property name: Indicates level of support of processor implementation specific options not described by the Processor Architecture and not supported on partitions that contain the **"ibm,migratable-partition"** property.

prop-encoded-array: One or more *raw-pi-attribute-descriptor*(s).

raw-pi-attribute-descriptor: Consists of a *raw-pi-attribute-header* immediately followed by a *raw-pi-attribute-specifier.*

raw-pi-attribute-header: Consists of two bytes. The first byte is an unsigned integer representing a value from 1 to 254. The first byte specifies the number of bytes implemented by the platform of the *raw-pi-attribute-specifier.* The second byte is an unsigned integer specifying the *raw-pi-attribute-specifier-type*.

raw-pi-attribute-specifier: The *raw-pi-attribute-specifier* is defined by the *raw-pi-attribute-specifier-type* of the *raw-pi-attribute-header.* The *raw-pi-attribute-specifier* for the *raw-pi-attribute-specifier-type* value of 0 is defined by Table 262, "raw-pi-attribute-specifier definition for raw-pi-attribute-specifier-type value 0," on page 771.

Table 262. *raw-pi-attribute-specifier* definition for *raw-pi-attribute-specifier-type* value 0

Byte Number	Bit Number	Attribute Name	Description
0	0	FPR GPR Move Instructions	The value of 1 indicates that the PA processor defined by this CPU node implements the *mftgpr* and *mffgpr* instructions as described by IBM *POWER6® CEC Book IV Implementation Features*; else not supported.
	1-7	Reserved	Reserved bits within defined bytes *shall* be zero.
1-255	0-7	Reserved	Reserved bits within defined bytes *shall* be zero.

"ibm,pa-optimizations"

property name: Indicates the level of support of performance variabilities described by the Processor Architecture.

prop-encoded-array: One or more *pa-optimization-attribute-descriptor*(s).

pa-optimization-attribute-descriptor: Consists of a *pa-optimization-attribute-header* immediately followed by a *pa-optimization-attribute-specifier.*

pa-optimization-attribute-header: Consists of two bytes. The first byte is an unsigned integer representing a value from 1 to 254. The first byte specifies the number of bytes implemented by the platform of the *pa-optimization-attribute-specifier.* The second byte is an unsigned integer specifying the *pa-optimization-attribute-specifier-type*.

pa-optimization-attribute-specifier: The *pa-optimization-attribute-specifier* is defined by the *pa-optimization-attribute-specifier-type* of the *pa-optimization-attribute-header.* The *pa-optimization-attribute-specifier* for the *pa-optimization-attribute-specifier-type* value of 0 is defined by Table 263, "pa-optimization-attribute-specifier definition for pa-optimization-attribute-specifier-type value 0," on page 771.

Table 263. *pa-optimization-attribute-specifier* definition for *pa-optimization-attribute-specifier-type* value 0

Byte Number	Bit Number	Attribute Name	Description
0	0-7	Stream IDs	The value is an unsigned quantity indicating the number of data stream IDs supported. The value of this byte *shall* be zero for processors that do not support data streams.

Table 263. *pa-optimization-attribute-specifier* definition for *pa-optimization-attribute-specifier-type* value 0 ***(Continued)***

Byte Number	Bit Number	Attribute Name	Description
1	0-7	Default Prefetch Depth	The value in the Default Prefetch Depth (DPFD) field of the Logical Partitioning Control Register (LPCR) as described by version 2.05 of PA. Unimplemented high order bits *shall* be zero. This byte is valid only if the "2.05 Data Stream Support" bit of **"ibm,pa-features"** is set to one; else this byte is undefined.
2-255	0-7	Reserved	Reserved bits within defined bytes *shall* be zero.

C.6.1.5 TLB properties

Since the PA defines the MMU as being part of the processor, the properties defined by Section 3.6.5 of *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2] and the following MMU-related properties shall be presented under "cpu" nodes.

"tlb-size"

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

"tlb-sets"

Standard *property name*, 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.

"tlb-split"

This property, if present, shall indicate that the TLB has a split organization. The absence of this property shall indicate that the TLB has a unified organization.

"d-tlb-size"

Standard *property name*, encoded as with **encode-int**, that represents the total number of d-TLB entries.

"d-tlb-sets"

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

"i-tlb-size"

Standard *property name*, encoded as with **encode-int**, that represents the total number of i-TLB entries.

"i-tlb-sets"

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

"tlbia"

prop-encoded-array: <none>

This property, if present, indicates that the PA processor defined by this CPU node implements the *tlbia* instruction. The absence of this property indicates that the PA processor defined by this CPU node does not support the *tlbia* instruction.

C.6.1.6 Internal (L1) cache properties

The PA defines a Harvard-style cache architecture; however, unified caches are an implementation option. All of the PA cache instructions act upon a cache "block". The coherence block size, if different from the cache block size, is reported via the **"i-cache-line-size"** and **"d-cache-line-size"** properties. The internal (also referred to as "L1") caches of PA processors are represented in the OF device tree by the following properties contained under **"cpu"** nodes.

"cache-unified"

This property, if present, indicates that the internal cache has a physically unified organization. Absence of this property indicates that the internal caches are implemented as separate instruction and data caches. Unless otherwise noted, separate instruction and data caches require the architected instruction sequence for instruction modification so that data cache stores appear in the instruction cache.

"i-cache-size"

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

"i-cache-sets"

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

"d-cache-size"

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

"d-cache-sets"

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

"l2-cache"

Standard *property name*, encoded as with **encode-int**, that represents the next 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 next level of cache.

"i-cache-line-size"

Standard *property name*, encoded as with **encode-int**, that represents the internal instruction cache's coherency block size (line size), in bytes, if different than its cache block size.

"d-cache-line-size"

Standard *property name*, encoded as with **encode-int**, that represents the internal data cache's coherency block size (line size), in bytes, if different than its cache block size.

Note: If this is a unified cache, the corresponding i- and d- sizes must be equal.

C.6.1.7 Memory Management Unit properties

To aid a client in "taking over" the translation mechanism and still interact with OF (via the client interface), the client needs to know what translations have been established by OF. The following standard property shall exist within the package to which the "mmu" property of the /chosen package refers.

"translations"

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

(virt size phys mode)

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

C.6.1.8 SLB properties

Since the PA defines the MMU as being part of the processor, the properties defined by Section 3.6.5 of *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2] and the following MMU-related properties as appropriate to the specific processor implementation shall be presented under "cpu" nodes.

"slb-size"

Standard *property name*, encoded as with **encode-int**, that represents the total number of SLB entries.

Note: *Power ISA* [1] requires that the SLB be fully-associative, and appear to be a unified organization. Therefore, properties to report SLB sets, split, and sizes and sets of i and d SLBs are not defined.

C.6.2 Ancillary (L2,L3...) cache node properties

Some systems might include secondary (L2) or tertiary (L3), etc. cache(s). As with the L1 caches, they can be implemented as either Harvard-style or unified. Unlike the L1 properties, that are contained within the **"cpu"** nodes, the properties of ancillary caches are contained within other device tree nodes.

The following properties define the characteristics of such ancillary caches. These properties shall be contained within a child node of one of the CPU nodes or, for platforms that support dynamic reconfiguration of cpus, the CPUS node; this is to allow path-name access to the node. These properties shall always be contained within a child node of the CPUS node. All **"cpu"** nodes that share the same ancillary cache (including the cpu node under which the ancillary cache node is contained) shall contain an **"l2-cache"** property whose value is the **phandle** of that ancillary cache node.

Note: The **"l2-cache"** property shall be used in one level of the cache hierarchy to represent the next level. The device node for a subsequent level shall appear as a child of one of the caches in the hierarchy to allow path-name traversal. The preceding sentence does not apply to platforms that support dynamic reconfiguration of cpus or platforms designed after 07/2005.

"device_type"

Standard *property name*; the device_type of ancillary cache nodes shall be **"cache"**.

"cache-unified"

This property, if present, indicates that the cache at this node has a physically unified organization. Absence of this property indicates that the caches at this node are implemented as separate instruction and data caches. Unless otherwise noted, separate instruction and data caches require the architected instruction sequence for instruction modification so that data cache stores appear in the instruction cache.

"i-cache-size"

Standard *property name*, encoded as with **encode-int**, that represents the total size (in bytes) of the instruction cache at this node.

"i-cache-sets"

Standard *property name*, encoded as with **encode-int**, that represents number of associativity sets of the instruction cache at this node. A value of 1 signifies that the instruction cache is fully associative.

"d-cache-size"

Standard *property name*, encoded as with **encode-int**, that represents the total size (in bytes) of the data cache at this node.

"d-cache-sets"

Standard *property name*, encoded as with **encode-int**, that represents number of associativity sets of the instruction cache at this node. A value of 1 signifies that the instruction cache is fully associative.

"l2-cache"

Standard *property name*, encoded as with **encode-int**, that represents the next 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-cache-line-size"

Standard *property name*, encoded as with **encode-int**, that represents the internal instruction cache's line size, in bytes, if different than its block size.

"d-cache-line-size"

Standard *property name*, encoded as with **encode-int**, that represents the internal data cache's line size, in bytes, if different than its block size.

Note: If this is a unified cache, the corresponding i- and d- sizes must be equal.

C.7 Methods

This section describes the additional standard methods required of a PA OF implementation.

C.7.1 MMU related methods

The MMU methods defined by section 3.6.5. of *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2] shall be implemented by CPU nodes. The value of the **mode** parameter for the relevant methods (e.g., **map**) shall 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 re-

served bit that shall be 0. In order for I/O accesses to be properly performed in a LoPAPR system, address ranges that are mapped by **`map-in`** shall be marked as Cache-Inhibited, Guarded.

The default mode (i.e., the mode specified when the value of the *mode* argument is -1) for the **`map-in`** and **`modify`** MMU methods of CPU nodes is defined as follows:

If the beginning of the physical address range affected by the operation refers to system memory, the values for **`WIMGxPP`** shall be W=0, I=0, M=0, G=1, PP=10.

If the beginning of the physical address range affected by the operation refers to an I/O address, the values for WIMGxPP shall be W=1, I=1, M=0, G=1, PP=10.

C.8 Client Interface Requirements

A PA OF implementation shall implement a client interface (as defined in chapter 6 of *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2]) according to the specifications contained within this section.

C.8.1 Calling Conventions

To invoke a client interface service, a *client program* constructs a client interface *argument array* as specified in the core OF 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 executing a **`bc-trl`**.)

The term “preserved” below shall mean that the register has the same value when returning as it did when the call was made.

Table 264. Register usage conventions

Register(s)	Value -- real-mode		Value -- virt-mode		Notes
	If either the FWNMI, or LPAR option is implemented	If neither the FWNMI or LPAR option is implemented	If either the FWNMI, or LPAR option is implemented	If neither the FWNMI or LPAR option is implemented	
msr	client interface shall preserve	client interface shall preserve	same as real-mode	client interface shall not modify	
cr	client interface shall preserve	client interface shall preserve	same as real-mode	same as real-mode	1
r1-r2	client interface shall preserve	client interface shall preserve	same as real-mode	same as real-mode	
r3	argument array address on client interface entry	argument array address on client interface entry	same as real-mode	same as real-mode	2
	result value (true or false) on client interface return	result value (true or false) on client interface return	same as real-mode	same as real-mode	2
r13-r31	client interface shall preserve	client interface shall preserve	same as real-mode	same as real-mode	
sprg0, sprg1, and sprg3	client interface shall preserve	client interface shall not modify	same as real-mode	same as real-mode	

Table 264. Register usage conventions ***(Continued)***

Register(s)	Value -- real-mode		Value -- virt-mode		Notes
fpscr	client interface shall preserve	client interface shall preserve	same as real-mode	same as real-mode	
f0-f31	client interface shall preserve	client interface shall preserve	same as real-mode	same as real-mode	
lr, ctr, xer	undefined	undefined	same as real-mode	same as real-mode	
sr0-sr15	client interface shall preserve	client interface shall preserve	same as real-mode	client interface shall not modify	
vr0-vr31	client interface shall preserve	client interface shall preserve	same as real-mode	same as real mode	
dec	client interface shall preserve	client interface shall not modify	same as real-mode	same as real mode	
Other SPRs	client interface shall preserve	client interface shall preserve	same as real-mode	same as real-mode	3

Notes:

1. Only the non-volatile fields (**`cr2-cr4`**) need to be preserved.
2. As defined by section 6.3.1. of *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2].
3. Other special purpose registers

The *client interface handler* shall 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* shall 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. OF shall *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`**.

C.9 Client Program Requirements

C.9.1 Load Address

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

C.9.2 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.

C.9.2.1 Initial Register Values

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

Table 265. 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 C.9.4, “Interrupts,” on page 779	
	IR,DR, see Section C.5.2.1, “Real-Mode,” on page 756	
	SF=0, 32-bit mode	
	ILE,LE, little endian support	2
r1	See Section C.9.2.2, “Initial Stack,” on page 779	
r2	0	3
r3	reserved for platform binding	4
r4	reserved for platform binding	4
r5	See Section C.9.2.3, “Client Interface Handler Address,” on page 779	
r6, r7	See Section C.9.2.4, “Client Program Arguments,” on page 779	
Other user mode registers	0	

Notes:

1. OF will typically require the use of external interrupts for its *user interface*. However, when a *client program* is invoked, external interrupts shall 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. OF 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 the relevant section of the platform binding.

C.9.2.2 Initial Stack

Client programs shall be invoked with a valid stack pointer (**r1**) with at least 32 KB of memory available for stack growth. The stack pointer shall 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 address; if in Virtual-Mode, the address in r1 is a mapped virtual address.

C.9.2.3 Client Interface Handler Address

When client programs are invoked, **r5** shall 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 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.

C.9.2.4 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 shall be passed in registers **r6** and **r7**, respectively. For programs booted directly by OF, 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 OF standard makes no provision for specifying such an array or its contents. Therefore, in the absence of implementation-dependent extensions, a client program executed directly from an OF implementation will not be passed such an array. However, intermediate boot programs that simulate or propagate the OF 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 OS options, are provided to client programs via the **"bootargs"** and **"bootpath"** properties of the **/chosen** node.

C.9.3 Caching

The caches of the processor shall be enabled when the client program is called. The I-cache shall be consistent with the D-cache for all memory areas occupied by the client program. Memory areas allocated on behalf of the client program shall be marked as cacheable. Accesses to "I/O" devices (especially, to devices across "bridges") shall be made with the register access words (e.g., **%rl@**). All processors in a SMP system shall have the same consistent view of all memory areas (for data references). No more than one processor shall have a modified copy of the same data area in its cache when the client program is called.

Note: If firmware makes cacheable M=0 data references from different processors on a SMP system, it may have to perform additional cache management to meet this requirement.

C.9.4 Interrupts

OF 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, OF does not assume ownership of interrupts.

Note: There used to be a paragraph here that said an area of memory was to be reserved by the client program for the exclusive use of OF. This requirement has been removed, since the sharing of interrupt vectors on these platforms has not been found to be practical.

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

C.9.5 Client callbacks

This section defines the callback mechanism that allows OF to access services exported to it by the client program. As described in section 6.3.2 and the glossary entries for **`callback`** and **`$callback`** in *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], the callback mechanism follows the same rules as those of Client interface calls. I.e., an *argument array* is constructed by OF 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 OF by means of the **`set-callback`** client call.

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

C.9.5.1 Real-Mode physical memory management assist callback

Once the control of physical memory is transferred to the client program, OF which is running in real-mode shall use the callback service provided by the client program to allocate physical memory. Client programs which expect OF to operate in read-mode must implement the following physical memory management client callback routines for OF:

`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 C.7, "Methods," on page 775 A non-zero error code shall be returned if the mapping cannot be performed. If error code is zero (i.e. allocation is succeeded) the routine returns the base address of the physical memory allocated for OF.

C.9.5.2 Virtual address translation assist callbacks

As mentioned in Section C.5.2.6, "Client Interface (Virtual-Mode)," on page 758, when OF is 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 OF specification *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2].

`map`

IN: [address] phys, [address] virt, size, mode

OUT: throw-code, 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 shall be placed in the *error* cell of the argument array; a non-zero error code shall be returned in *error* if the mapping cannot be performed.

unmap

IN: [address] virt, size

OUT: throw-code

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

translate

IN: [address] virt

OUT: throw-code, error, [address] real, 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 shall be placed into the HTAB for this translation, the number of return cells shall be four with the resulting real address returned in *real* and *error* shall be set to **false** (0). If the translation is not successful, the number of return cells shall be two and *error* shall be set to a non-zero error code.

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

C.10 User Interface Requirements

An implementation of OF for the PA shall conform to the core requirements as specified in *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2] and the following PA-specific extensions.

C.10.1 Machine Register Access

The following *user interface* commands represent PA registers within the *saved program state*. Executing the command returns the saved value of the corresponding register. The saved value may be set by preceding the command with **to**; the actual registers are restored to the saved values when **go** is executed.

The following command displays the PA processor's *saved program state*.

.registers

C.10.1.1 Branch Unit Registers

%cr

Access saved copy of Condition Register.

%ctr

Access saved copy of Count Register.

%lr

Access saved copy of Link Register.

%msr

Access saved copy of the low order 16 bits of SRR1 register.

%srr0 and **%srr1**

Access saved copy of Save/Restore Registers.

%pc

An alias of "**%srr0**"

C.10.1.2 Fixed-Point Registers

%r0 through **%r31**

Access saved copies of fixed-point registers.

%xer

Access saved copy of XER register.

%sprg0 through **%sprg3**

Access saved copies of SPRG registers.

C.10.1.3 Floating-Point Registers

Unlike the other registers, the floating point unit registers are not normally saved, since they are not used by OF. The following access words, therefore, access the registers directly.

%f0 through **%f31**

Access floating point registers.

%fpscr

Access Floating Point Status and Control Register.

C.11 Configuration Variables

In addition to the standard Configuration Variables defined by the core OF document *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [2], the following Configuration Variables shall be implemented for the PA:

"little-endian?"

This boolean variable controls the endian-mode of OF. If **true** (-1), the endian-mode is Little-Endian; if **false** (0), the endian-mode is Big-Endian. The default value is implementation dependent.

"real-mode?"

This boolean variable controls the address translation mode of OF. If `true` (-1), the addressing mode is Real-Mode; if `false` (0), the addressing mode is Virtual-Mode. The default value is implementation dependent.

"real-base"

This integer variable defines the starting physical address to be used by OF.

"real-size"

This integer variable defines the size of the physical address space which can be used by OF.

"virt-base"

This integer variable defines the starting virtual memory address which can be used by OF.

"virt-size"

This integer variable defines the size of the virtual address space which can be used by OF.

"load-base"

This integer variable defines the default load address for *client program*s when using the **load** method. The default value is implementation dependent.

C.12 MP Extensions

This section specifies the application of OF to PA multiprocessor (MP) systems. An OF implementation for an MP PA system shall implement the extensions described in this section as well as the requirements described in previous sections of this binding.

C.12.1 The Device Tree

This section defines an additional property under the **/chosen** node for a MP extension. Refer to Section C.6.1.1, "The Device Tree," on page 759 for more details about the device tree structure for a MP Configuration.

C.12.1.1 Additional Properties

/chosen Node Properties

"cpu"

property name, identifies the running CPU.

prop-encode-array: An integer, encoded as with **encode-int**, which contains the i-handle of the CPU node that is associated with the "running" CPU.

C.12.2 Initialization

OF shall select one processor, using an algorithm of its choice, to be the "master" processor, which performs the role of the single processor on a uniprocessor system, either booting the client or providing the user interface. OF shall place all the remaining processors into stopped state, a state in which the processor does not perform OF or client functions and does not interfere with the operation of the master processor. A processor in stopped state remains in that state unless and until an executing client starts it using the **start-cpu** client service defined below.

Client programs shall use the OF **start-cpu** client interface service to start all processors before it reclaims the OF memory

On machines in which a machine check on one processor is broadcast to all processors, the processors which are either in the idle or stopped state shall not change their states due to a machine check on another processor: OF shall not depend on the contents of the low vector (IP=0) in the event of a machine check.

Figure 37, "Stopped, Running, and Idle State Diagram," on page 784 depicts the relationship of the Running, Stopped and Idle States to each other. The *Client Interface Service* calls are shown as how to move between the states.

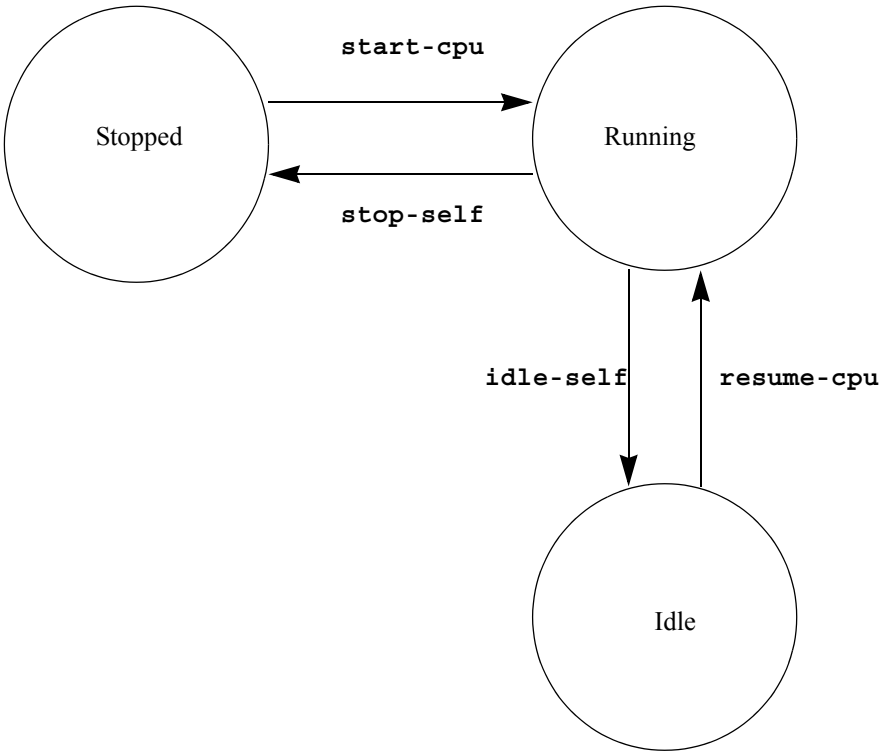


Figure 37. Stopped, Running, and Idle State Diagram

Note: OF's memory cannot be reclaimed by a client if a CPU is in the “stopped” or “idle” state.

C.12.3 Client Interface Services

The following client interface services are added for MP support on PA systems. These interfaces make the client program responsible for any Inter-CPU communication needed for these interfaces. The rationale for this is to architecturally separate the Inter-CPU communication mechanism of the firmware from the client program and vice versa.

`start-cpu`

IN: nodeid, pc, arg

OUT: none

This client interface service starts the CPU. The *nodeid* is the phandle of a node whose device_type is “cpu”.

`Start-cpu` arranges for the CPU identified by phandle in *nodeid* to begin executing client code at the real address given by the *pc* input with an argument, *arg,* passed in register r3. When it begins execution, the started processor shall be in the endian mode of the client program, and in real (not translated) addressing mode. The contents of registers other than r3 are indeterminate.

A client should not call **`start-cpu`** for the processor on which it is running, effectively restarting with a new pc and abandoning the only client thread. A jump or branch instruction shall be used instead to achieve the objective.

start-cpu permits more than one processor to run at the same time, enabling multi-threaded client execution. In general, an OF client shall avoid multi-threaded operation within OF. Usually, this means that client threads running on different CPUs must use mutual exclusion to prevent more than one processor from making client service requests at any one time. The exceptions are that a client thread may invoke either the **stop-self** or **idle-self** client services defined below at any time.

Note: The results are undefined if the CPU identified by *phandle* has already been started (e.g. it is already running and has not exited) or *phandle* is not a valid package handle of a CPU device node.

stop-self

IN: none

OUT: none

OF places the processor on which the caller was running into the “stopped” state. The client program is not-resumable.

Note: When an MP client program exits, one CPU invokes the **exit** client interface service, the others invoke the **stop-self** service.

idle-self

IN: none

OUT: none

OF places the CPU on which this service was invoked into an 'idle' state, saving the *current state* of the client program, so that the client program may be resumed.

A processor in idle state can be resumed using **resume-cpu** service defined below or restarted using **start-cpu**. If the processor is resumed, it executes a normal return to the client, as if its call to **idle-self** had just completed.

Note: When a client program wants to enter the firmware user interface, one CPU invokes the **enter** client interface service, the others invoke the **idle-self** service. The rational is that the user interface may affect the machine state in any way that it desires, therefore the client shall not depend on it.

resume-cpu

IN: nodeid

OUT: none

This client interface service is used to resume an *idled* CPU. The *nodeid* is the phandle of a CPU node in idle state.

resume-cpu arranges for that CPU to restore the CPU’s state as saved by **idle-self** and begin return to the client, completing the idle-self client service call that placed the CPU into idle state. The results are undefined if the CPU identified by *phandle* is not in an *idle* state by a previous call to the **idle-self** client interface service.

Note: When the client program is resumed via the GO (or similar) user interface command, the client program is resumed on the CPU which called the **`enter`** service; the client program is responsible for calling the **`resume-cpu`** service to resume other idled CPU's, if that is the desired client program behavior.

C.12.4 Breakpoints

If the breakpoint is taken by the firmware, without the client program's assistance, the other CPUs will continue to run in the client program. The client program may field the breakpoint 'trap' or 'vector' and idle the other CPUs before entering the PROM. The platform binding document has to specify how this is done to avoid loss of state at breakpoint time.

C.12.5 Serialization

The firmware is a single threaded program, from the client program's point of view. Only the **`idle-self`**, **`stop-self`**, **`enter`** and **`exit`** client interfaces may be invoked simultaneously on different CPUs. Furthermore, only a single CPU may invoke the **`enter`** or **`exit`** client interface at any one time. The other CPUs must use the **`idle-self`** or **`stop-self`** client interface service.

Note: The results are undefined if the client program invokes client interface services including breakpoint traps (other than the **`enter/exit stop-self/idle-self`** case listed above) simultaneously on more than a single CPU.

Note: Since locking mechanisms are subject to client program policy, the client program is responsible for implementing any necessary mechanism to insure that it adheres to this policy. Further, the client program should disable any preemption mechanism before calling a client interface service to avoid rescheduling a thread of execution in the firmware on a different CPU if such a mechanism exists in the client program.

D A Protocol for a Virtual TTY Interface

D.1 Overview

This appendix defines a protocol to support partition use of a physical serial port using a virtual TTY (VTY) interface. A protocol is required to send control and status information of the physical device using a data only transport.

Specifically, this protocol is used to allow the Virtual Terminal (VTERM) option, as defined in Section 16.6, "Virtual Terminal (Vterm)," on page 582, as the interface to communicate with a physical serial port which is under control of the platform instead of the OS.

This appendix describes a connection between the platform, which has physical control of the serial port and is an endpoint of the VTERM interface, and a partition, which is the other endpoint of the VTERM interface. The protocol described here provides a means to communicate both data and control information over this data-only interface.

D.2 Protocol Definition

D.2.1 Packet Formation

All information is sent in packets. There are four types of packets supported for version 0, data packets, control packets, query packets and query response packets. A packet consists of a one byte header, which defines the type of packet, followed by a one byte length field, followed by a two byte sequence number, followed by the packet payload, which depends on the type of packet. This means that the minimum packet size is 5 bytes, in the case of a one-byte data packet.

The one-byte length field for the packet is the length of the entire packet, including the header byte and the length field itself.

On the partition side the H_GET_TERM_CHAR and H_PUT_TERM_CHAR hypervisor calls are used to read and write data. These HCALLS read and write up to 16 bytes of data at once. There is no relationship between the data in a particular HCALL and packet boundaries. A single packet can span multiple HCALLS, and a single HCALL can contain data from more than one packet.

For version 0, the following packet headers are defined:

D.2.1.1 Data Packet

#define VS_DATA_PACKET_HEADER 0xFF

This packet type is used to send data.

The data packet is defined in Table 266, "VTERM Data Packet," on page 788.

Table 266. VTERM Data Packet

Packet Offset	Member Size (Bytes)	Description
0x0	0x1	Packet Header (0xFF)
0x1	0x1	Total size of packet in bytes, including this header
0x2	0x2	Sequence Number (see description below)
0x4	variable	Data

D.2.1.2 Control Packet

#define VS_CONTROL_PACKET_HEADER 0xFE

This packet type is used to send commands that control the operation of software or hardware on the other side of the link, and to send notification of status changes on the other side.

The control packet is defined in Table 267, “VTERM Control Packet,” on page 788.

Table 267. VTERM Control Packet

Packet Offset	Member Size (Bytes)	Description
0x0	0x1	Packet Header (0xFE)
0x1	0x1	Total size of packet in bytes, including this header
0x2	0x2	Sequence Number (see description below)
0x4	0x2	Control verb (see description below)
0x6	optional and variable	Depending on verb, further data

The following control verbs are supported.

D.2.1.2.1 VSV_SET_MODEM_CTL Verb (0x01)

Protocol Version: 0x00

Note: One-way. This verb is only sent by the partition to the platform.

Data Member Definition: The data member for this verb consists of 8 bytes, as defined in Table 268, “VSV_SET_MODEM_CTL Verb Data Member,” on page 788.

Table 268. VSV_SET_MODEM_CTL Verb Data Member

Packet Offset	Member Size (Bytes)	Description
0x6	0x4	VS_MODEM_CTL word (defined below)
0xA	0x4	VS_MODEM_CTL mask, defining which bits in the control word above are to be updated.

VS_MODEM_CTL is a 4 byte bit-mask. The following bits are defined. Note that each bit position has an associated protocol version. Note that some bits can bet set (noted as R/W in the table), others only can be read (noted as R/O in the table). This verb will set only bits marked R/W.

The update of the serial port hardware driven by this command must be serialized with data packets.

In the data portion of the packet, the first word defines the bit values to be set; the second word is a mask defining which bits are to be updated. See Table 269, “VS_MODEM_CTL Bit Definition,” on page 789.

Table 269. VS_MODEM_CTL Bit Definition

Bit Position	Protocol Version Supported	R/O, R/W	Name	Description
0x0000001	0x00	R/W	TSDTR	Data Terminal Ready
0x0000020	0x00	R/O	TSCD	Carrier Detect

D.2.1.2.2 VSV_MODEM_CTL_UPDATE Verb (0x02)

Protocol Version: 0x00

Note: One-way. This verb is only sent by the platform to the partition.

Data Member Definition: The data member for this verb consists of 4 bytes, as defined in Table 270, “VSV_MODEM_CTL_UPDATE Verb Data Member,” on page 789

Table 270. VSV_MODEM_CTL_UPDATE Verb Data Member

Packet Offset	Member Size (Bytes)	Description
0x6	0x4	VS_MODEM_CTL word (defined above)

This packet is sent by platform to the partition to inform the partition of a change in status of certain bits in the VS_MODEM_CTL word. The bits which cause this command to be sent when they transition are defined in Table 271, “VS_MODEM_CTL Word Bits,” on page 789. This command should be serialized with data packets. The protocol version in Table 271, “VS_MODEM_CTL Word Bits,” on page 789 defines the first (lowest) version of the protocol in which a transition of this bit should cause the command to be sent.

Table 271. VS_MODEM_CTL Word Bits

Bit Name	Protocol Version Supported	Description
TSCD	0x00	A transition of carrier detect must cause this packet to be sent.

D.2.1.2.3 VSV_RENEGOTIATE_CONNECTION Verb (0x03)

Protocol Version: 0x00

Data Member Definition: No data member for this verb.

This verb forces the protocol into “closed” state; see the Connection Negotiation section below for the definition and meaning of this state.

In normal operation it is expected that this command will be send from the partition to the platform. However, if the platform is aware that it will be unable to continue for some reason for a finite period of time, it can send this command to the partition.

The platform may send this command even if the protocol is not in an open state; this is to allow the platform to unconditionally close the platform during an error recovery processor.

When the partition transitions control of the VTY between independent entities, such as when it is given from OF control the to the OS, the protocol should be closed with this command by the component that is relinquishing it and reopened by the receiving component.

D.2.1.3 Query Packet

#define VS_QUERY_PACKET_HEADER 0xFD

This packet is used to send queries to the other side of the link. The other side responds by sending a query response packet, defined below. There is in some case implied control of the state of the other side of the link, as a series of queries and responses are used to initialize (or re-initialize) the protocol.

The query packet is defined in Table 272, “VTERM Query Packet,” on page 790.

Table 272. VTERM Query Packet

Packet Offset	Member Size (Bytes)	Description
0x0	0x1	Packet Header (0xFD)
0x1	0x1	Total size of packet in bytes, including this header (0x6)
0x2	0x2	Sequence Number (see description below)
0x4	0x2	Query verb (see description below)

The following query verbs are supported:

D.2.1.3.1 VSV_SEND_VERSION_NUMBER Verb (0x01)

Protocol Version: 0x00

Response: The query response data member will contain the one-byte version number of the highest version of the protocol supported by the driver.

D.2.1.3.2 VSV_SEND_MODEM_CTL_STATUS Verb (0x02)

Protocol Version: 0x00

Response: The query response data member will contain the four-byte VS_MODEM_CTL word defined above in the VSV_SET_MODEM_CTL verb.

Note: One-way. This verb is only sent by the partition to the platform.

D.2.1.4 Query Response Packet

#define VS_QUERY_RESPONSE_PACKET_HEADER 0xFC

This packet is used to reply to query packets sent from the other side of the link, and are sent only in response to query packets.

The query response packet is defined in Table 273, “VTERM Query Response Packet,” on page 791.

Table 273. VTERM Query Response Packet

Packet Offset	Member Size (Bytes)	Description
0x0	0x1	Packet Header (0xFC)
0x1	0x1	Total size of packet in bytes, including this header
0x2	0x2	Sequence Number (see description below)
0x4	0x2	Query verb -- matches the query verb for which this is a response
0x6	0x2	Query Sequence Number -- the sequence number of the query packet for which this is a response
0x8	variable	Verb-specific response data

D.2.2 Verb Formation

A verb, either for a command or query packet, consists of two bytes. The first byte is a version of the protocol associated with the verb, and the second byte is the verb itself. This allows the flexibility to redefine or add function to verbs in the future. More importantly, it allows partners at either end of the protocol to find a “least common denominator” at which they can work.

The connection sequence for partners to begin communication (defined below) causes each side of the protocol to learn the highest level of the protocol supported by the party on the other side. If one party discover that the party on the other end supports a lower version of the protocol, it is expected that only verbs at the lower version of the protocol will be used. If an unknown verb is received, the command is discarded without response.

D.2.3 Sequence Numbers

Each packet has a sequence number. Sequence numbers start at 0 and increment by one with each packet sent. There are separate name spaces for sequence numbers in each direction, but all packets of all types in a direction are in the same sequence number name space. The sequence number is unsigned, and the number following 0xFFFF is 0x0. Sequence numbers are used to match query packets with query response packets, and for general debugging.

The sequence numbers in data packets represent a fair amount of bandwidth overhead, especially if the number of bytes of data per packet is small in practice. If the performance of this facility is found to be inadequate due to bandwidth reasons, developers on both sides should recognize that sequence numbers for data packets represents the “low hanging fruit” to fix that problem, so are cautioned not to create logical dependencies on sequence numbers for data packets. For other packet types logical dependencies are acceptable, and indeed are built into the protocol definition.

Implementation Note: Developers should consider field problem determination in their designs. However, other than providing packet sequence numbers in the protocol, RAS is outside the scope of the protocol itself.

D.2.4 Flow Control

The partition side of the connection must understand what type or types of flow control are supported by the platform (physical) side of the connection. The options are no flow control, software flow control and hardware flow control.

In version 0 of this protocol it is implied that only software flow control is implemented by the platform side. If future platforms implement other options, a new version of this protocol must be created that includes a verb for the partition side to determine what type or types of flow control are supported, and to negotiate what type is to be used in the case that more than one is supported.

D.2.5 Packet Type and Verb Summary

A summary of packet types and verbs can be found in Table 274, “VTERM Packet Type and Verb Summary,” on page 792.

Table 274. VTERM Packet Type and Verb Summary

Packet ID	Verb ID	Lowest Version Supported	Description
0xFF	--	0	VS_DATA_PACKET_HEADER
0xFE	--	0	VS_CONTROL_PACKET_HEADER
0xFE	0x1	0	VSV_SET_MODEM_CTL
0xFE	0x2	0	VSV_MODEM_CTL_UPDATE
0xFE	0x3	0	VSV_RENEGOTIATE_CONNECTION
0xFE	0x3	0	VSV_CLOSE_PROTOCOL (alias for above)
0xFC	--	0	VS_QUERY_RESPONSE_PACKET_HEADER
0xFC	0x1	0	VSV_SEND_VERSION_NUMBER
0xFC	0x2	0	VSV_SEND_MODEM_CTL_STATUS
0xFD	--	0	VS_QUERY_RESPONSE_PACKET_HEADER

D.3 Connection Negotiation

This protocol itself has a state of being open or closed. The state is “closed” at partition boot time and remains closed until a connection negotiation is initiated by the partition device driver. Once open, the state remains “open” until a VSV_CLOSE_PROTOCOL command is sent by either side.

While closed, both parties should “listen” on the VTY interface. However, while closed, only query and query response packets should be acted on; any data or control packets received should be discarded.

The connection process is initiated by the partition side. The sequence is as follows:

- Both sides are in closed state. Both side are “listening” to the VTY.
- The partition sends the VSV_SEND_VERSION_NUMBER query.
- The partition continues listening, but discard any packet that is not a VSV_SEND_VERSION_NUMBER query response.
- The platform replies with the VSV_SEND_VERSION_NUMBER query response packet.
- The platform clears any pending data in the serial port hardware.

- The platform sends the VSV_SEND_VERSION_NUMBER query packet.
- The partition responds with the VSV_SEND_VERSION_NUMBER query response packet.

At this point the protocol is open; any data received in the serial hardware by the platform from this point should be put into data packets and sent to the partition; any data packets received from the partition should be sent out the serial hardware.

The platform should implement a time-out after sending a command that expects a response, and after the time-out log an error. For version 0 of the protocol an appropriate value would be seconds or minutes; at least 10 seconds. The next version of the protocol should consider a property in the device tree node to communicate an appropriate time-out value.

E A Protocol for VSCSI Communications

E.1 Introduction

The purpose of this Appendix is to define the protocol used by virtual SCSI (vscsi) client drivers and vscsi server drivers in sufficient detail to ensure compatibility between unlike operating systems implementing these features. The SCSI Architecture Model (SAM-2) defines the following simplified abstract model and terminology for a SCSI system.

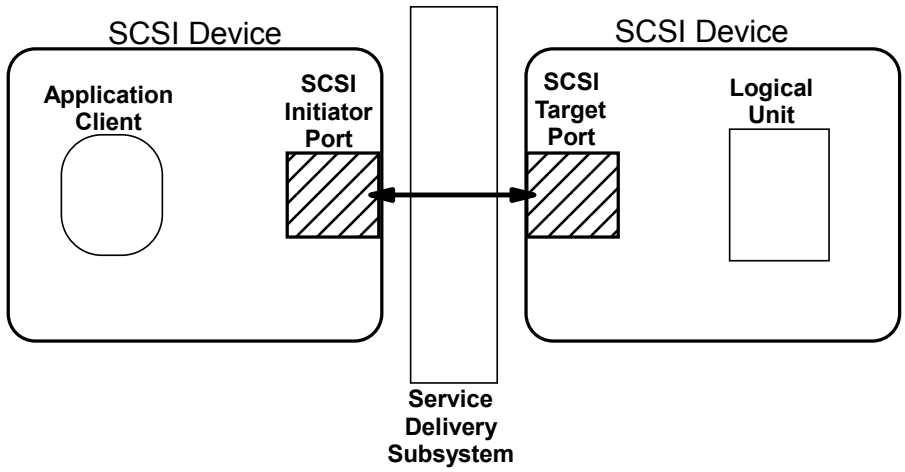


Figure 38. SCSI Initiator/Target Architecture

In Figure 38, “SCSI Initiator/Target Architecture,” on page 795, the Application Client is the application producing or consuming the data being stored. The SCSI Initiator Port is the virtual scsi client adapter running in the client partition. The Service Delivery System is the Hypervisor. The SCSI Target Port is the vscsi host (vhost) adapter running in the VIO server (VIOS). The Logical Unit is the entity providing the data storage services.

Note that the model is not symmetrical. Client adapters may communicate only with host adapters and host adapters may communicate only with client adapters. Each may communicate with a maximum of one partner at any point in time. Client adapters may exist only in client partitions. Host adapters may exist only in VIOSs. A client partition may have multiple client adapters and they may communicate with host adapters in the same or different VIOSs. A SCSI host adapter may have multiple Logical Units defined to it for use. Almost all messages are initiated by the client. The client and host adapters communicate using Command/Response Queues (CRQ) defined earlier in this document. A client may not read or write VIOS memory, it may only write to the VIOS CRQ. The VIOS may read and write to client partition memory, if the client passes the VIOS a DMA mapped address for that memory.

E.2 SCSI Remote DMA Protocol (SRP)

The protocol used for transferring data between the application client and the logical unit is the SCSI Remote DMA Protocol (SRP), revision 16.a, as defined by the InterNational Committee for Information Technology Standards (INCITS). Copies of the standard are available at the INCITS website at T10.org.

The client builds an SRP request in its address space, then DMA maps that request so that the VIOS can access it. The client notifies the VIOS of the request by including that mapped address in a CRQ message. A SCSI Command Data Block (CDB) is encapsulated within the SRP request. Also within the SRP request is a tag field, which is private to the client. The VIOS must not modify that tag value in any way. When the request is complete, the VIOS notifies the client of the completion by including that tag field in the CRQ message to the client. The client then uses that tag value to locate the request being completed.

If the SRP request expects to transfer any data, it also contains one of the two types of memory descriptors specified by the SRP standard, to describe the buffer(s) to be used in the data transfer. In the SRP memory descriptor, the virtual address field is the DMA mapped address of the buffer, to be used by the VIOS to transfer the data. The memory handle field is not used and should be initialized to zero.

Using the H_SEND_CRQ call, the client sends the SRP request to the VIOS. The first 64 bits of the message describe the type of message, the format, and the length. The second 64 bits of the CRQ message contain the DMA mapped address of the SRP request in the client partition memory. The H_SEND_CRQ call in the client generates a virtual interrupt in the VIOS, if the CRQ is going from empty to non-empty (edge-triggered interrupt).

The vhost driver uses the H_COPY_RDMA call and the mapped address to copy the SRP request from client partition memory into VIOS memory, examines the LUN to which the request is addressed, builds the appropriate structure to represent the request, according to the type of backing device, then queues the request to the backing device. The backing device may be an actual physical storage device, a software emulator, or some combination of device and emulation.

In the request is an SRP memory descriptor which contains one or more address/length pairs describing one or more buffers in client partition memory address space. The memory handle field of the SRP memory descriptor is not used by vscsi and should be initialized to zero. The virtual address field in the SRP memory descriptor is the DMA mapped address of a buffer in client partition memory that the backing device uses to transfer data. When the backing device services the request, it uses the same DMA services as it would to handle a request that had originated locally on the VIOS. However, DMA services on the VIOS use the H_COPY_RDMA call and the mapped address(es) in the SRP memory descriptor(s) to copy data directly between the client partition and the device, transparent to the device.

When the backing device has completed the request, it returns the request along with the results back to the VIOS driver. The VIOS driver builds an SRP response structure and copies that response back into client partition memory over the original SRP request. The SRP response includes any sense data that may have been returned with the request. All virtual devices are “auto-sense” devices. The vhost driver then notifies the client partition of the completed request by using H_SEND_CRQ to place a message in the client CRQ. The first 64 bits of the message describe the type, format, and length of the message. The second 64 bits are the “tag” field from the original SRP request. The client uses the tag to locate the SRP response and processes the response as appropriate.

It is important to note that the client partition must not unmap or modify in any way any of the memory associated with the request between the time that it notifies the VIOS of the request and the time that the VIOS notifies the client of the response.

E.3 Connection Establishment

Before any data can be transferred the two partitions have to establish a connection. Each partition is required to use H_REG_CRQ to register a Command/Response Queue (CRQ) with the Hypervisor to receive messages from the other partition. The size of the queue must be a multiple of 4KB. That memory must be DMA mapped. The size of the CRQ

merely determines the number of requests that a client may send to the VIOS in a single burst. The VIOS dequeues the requests as soon as it can, so in evenly balanced systems, where the VIOS has enough CPUs and memory to deal with all of its clients, the size of the CRQ is not a major limiting factor.

After H_REG_CRQ returns H_SUCCESS, each partition uses H_SEND_CRQ to attempt to send the Initialization message described previously in this document. This is a race condition that only one partition will win. The first partition to send the Initialization message receives an H_CLOSED return value from the Hypervisor, because the other partition has not yet registered its queue. The winning partition must wait to receive the Initialization message from its partner. The second partition to send the Initialization message receives an H_SUCCESS return value from the Hypervisor. That partition must wait for the Initialization Complete message from its partner. When a partition receives an Initialization message during connection establishment, it must respond with the Initialization Complete message and may then proceed to the next step. When a partition receives the Initialization Complete message during connection establishment, it may then proceed to the next step.

The next step in connection establishment is for the client to send one or more of the Management Datagrams (MAD) messages, described in detail later in this appendix. Since this is before the completion of the SRP Login request, no flow control has been established between the client and VIOS, so the client may send only one message at a time and must wait for the response from the VIOS before sending the next one. The exception is the optional MAD_EMPY_IU message. The client may follow that immediately with another message. The VIOS enforces flow control violations by logging and informative error, then closing and reopening the CRQ.

The client is required to send the MAD_ADAPTER_INFO_REQUEST. This provides the information that the VIOS displays with the lsmap command. The client may find it useful to save off and display the information that the VIOS returns in the response to the MAD_ADAPTER_INFO_REQUEST. Customers and service personnel frequently find this kind of information useful in unravelling some of the more elaborate configurations.

The client is required to send the MAD_CAPABILITIES_EXCHANGE if it wishes to participate in Partition Mobility operations. If it does not send this message, the VIOS does not consider it to be capable of being migrated.

If the client wishes to take advantage of the “fast fail” feature, it should send the MAD_ENABLE_FAST_FAIL message before the SRP login request.

The last step in connection establishment is the SRP login request. The Target Port Identifier field of the SRP Login request is not used by vscsi and should be initialized to zero. The client uses the SRP login request to specify the size of the largest SRP Information Unit that it will send to the VIOS and the format of the type of memory descriptors it intends to use. The size of the largest SRP Information Unit must also account for the size of the largest Management Datagram that the client expects to send. The VIOS may reject the SRP login if it cannot support the requested options. The VIOS will delay sending the response to the SRP login if it does not have any LUNs defined to it yet. This may be the case if both partitions are booted simultaneously and the VIOS has not completed the configuration process when the client sends the SRP login.

If the VIOS accepts the SRP login, it sends the SRP login response and notifies the client of this by placing the tag value from the SRP Login in the CRQ message. The request limit delta field of the SRP login response contains the maximum number of requests that the VIOS will allow the client to have active on the VIOS at any one time. This is the flow control mechanism. If the client violates this limit by sending too many requests, the VIOS will terminate the connection to the client. Note that each SRP response message also contains a request limit delta field. Typically, this is set to 1, to indicate that this completed request means another can be initiated. But if the VIOS has substantial resources added to it, it may increase the number of requests a client may have active, and will do so by setting a value greater than one in this field. Once the SRP login has been accepted, the VIOS may increase the number of requests, but it may never decrease that number until this connection is terminated.

After receiving an SRP Login Response for the VIOS, the client may then proceed with normal I/O data traffic. Usually, this starts with device discovery, where the client sends a REPORT_LUNS SCSI request to the VIOS. The VIOS responds with the list of LUNs that have been defined to this host adapter. The client may then use other SCSI requests to determine the identity and capabilities of each LUN.

If, after establishing a connection (VIOS sends SRP login response, and client receives it), a partition receives another Initialization message, Initialization Complete message, an SRP Login, or SRP Login response without some indication that the connection has been terminated, usually a Transport Event (described later), that is a protocol violation. Protocol violations are handled by logging an error, then closing and reopening the CRQ.

Likewise, after a connection has been terminated, the first messages must be either the Initialization or the Initialization Complete messages, as appropriate. Any other message is a protocol violation. And any SRP message received before a successful SRP Login is a protocol violation.

E.4 Connection Termination

A connection may be terminated by the client sending the VIOS an SRP_I_LOGOUT Information Unit. The VIOS may send the client an SRP_T_LOGOUT Information Unit, but only if the client has provided resources for this by sending the MAD EMPTY IU first. In the current implementation, neither is used and the drivers just call H_FREE_CRQ to terminate the connection.

A connection may also be terminated by the abnormal termination of a partition. When a partition crashes, the Hypervisor invalidates all of the memory mappings for that partition and places a Transport Event in the CRQ of the partner. If the partition that crashed was a client with requests active on the VIOS, when the storage drivers attempt to service those “in flight” requests, they find that the DMA mappings associated with the requests are no longer valid and usually will log one or more errors to that effect.

When a partition calls H_FREE_CRQ or crashes, the Hypervisor notifies the partner partition by placing a Transport Event in the partner’s CRQ. The first byte of the Transport Event is set to 0xFF, to indicate that this is a Transport Event. The second byte describes the event. A value of 0x01 indicates that the partner partition failed (crashed). A value of 0x02 indicates that the partner partition called H_FREE_CRQ. A value of 0x06 indicates to a client that it has been migrated. Only clients that send the MAD_CAPABILITIES message are candidates for being migrated. A VIOS cannot be migrated.

When a partition receives a Transport Event, it is not required to close its CRQ. It may instead just wait for an Initialization message from the partner partition when it is ready to communicate again.

E.5 Client Migration

When a client receives the migrated Transport Event, it must unmap any memory associated with any requests currently active on the VIOS. The client will never receive any completions for those requests and must remap and restart them at the end of the migration. Then the client must call H_ENABLE_CRQ until it returns H_SUCCESS. When the CRQ has been successfully enabled, the client sends the Initialization message and waits for the Initialization Complete message. It then goes through the rest of the connection establishment process, followed by the SRP login. After the VIOS sends the SRP Login response, the client may resume normal data transfers, starting with any requests that may have been active on the VIOS when the client was migrated.

Note that the partition identification information that the client sends in the MAD_ADAPTER_INFO message immediately after the migration event may be stale and reflect the identification of the original client partition before the migration. A client may register for DLPAR notification of migration, use that notification to obtain the current partition identification, and send another MAD_ADAPTER_INFO message to the VIOS with the correct information.

E.6 VSCSI Message Formats

All virtual scsi communications between client and server occurs using the Reliable Command/Response Transport and Logical Remote DMA functions defined earlier in this document. No other channels of communication are required to perform virtual SCSI functions.

These communications are made up of three classes of messages:

- Messages contained entirely within a single CRQ message
- SRP requests and responses, as defined by the SRP standard
- Management Datagrams

E.7 CRQ Message formats

CRQ messages are 16 bytes (128 bits) in length. Only the first byte is architected by the Reliable Command/Response Transport specification described earlier in this document. That specification is repeated in Table 275, “First Byte of the CRQ Message,” on page 799.

Table 275. First Byte of the CRQ Message

Value	Description
0x00	Element is unused -- all other bytes in the element are undefined
0x01 - 0x7F	Reserved
0x80	Valid Command/Response Entry -- the second byte defines the entry format
0x81-0xFE	Reserved
0xFF	Valid Transport Event -- the second byte defines the specific transport event

If the first byte of a CRQ message is 0x80, then it is a valid Command/Response entry and the second byte describes the format of message. Possible values for the second byte of the CRQ message when the first byte is 0x80 are shown in Table 276, “Second Byte of the CRQ Message,” on page 799.

Table 276. Second Byte of the CRQ Message

Format Byte Value	Definition
0x00	Unused
0x01	VSCSI SRP format
0x02	Management Datagram (MAD) format
0x03	i5os private format
0x04	AIX private format
0x05	Linux private format

Table 276. Second Byte of the CRQ Message ***(Continued)***

0x06	Message in CRQ format
0x07 - 0xFF	Reserved

If the format byte is 0x01, then the rest of the message is a vscsi SRP request or response message. The rest of the CRQ contents for this type of message is shown in Table 277, “CRQ VSCSI Client Message,” on page 800, for messages from the clients, and Table 278, “CRQ VSCSI VIOS Message,” on page 801, for messages from the VIOS. Messages with a format byte of 0x02 are Management Datagram messages, defined later in this Appendix. Messages formats of 0x03, 0x04, and 0x05 are reserved for private, Operating System-specific messages, and are currently unused by this implementation. Messages with a format byte of 0x06 are messages contained entirely within the CRQ.

E.8 CRQ VSCSI Client Message Format

Client messages are sent from the client partitions to the VIOS. Table 277, “CRQ VSCSI Client Message,” on page 800 shows the format of these messages,

Table 277. CRQ VSCSI Client Message

Byte Offset	0	1	2	3	4	5	6	7
0x00	CRQ Valid	CRQ Format	Reserved		Timeout		IU Length	
0x08	IU Data Pointer (TCE)							

For this type of message, the first byte (CRQ Valid) must be 0x80, and the second byte (CRQ Format) must be 0x01. Bytes 6 and 7 of the first long word are the IU Length, the length in bytes of the SRP Information Unit being passed. The second long word, IU Data Pointer, is the DMA mapped address of the SRP Information Unit being passed, typically an SRP Request. The VIOS uses the IU length and IU Data Pointer to copy the SRP Request into VIOS local memory for interpretation and processing.

Bytes 4 and 5 of the first long word, Timeout, are an optional suggested timeout value for this request. If this value is greater than zero, then the value may be passed along to the backing device as a suggestion for how long this request is expected to take to complete. The VIOS does not enforce any timeout values, but relies upon the underlaying backing devices.

Management Datagram (MAD) messages also use this same format, with the exception that the second byte (CRQ Format) must be set to 0x02. Bytes 6 and 7 of the first long word are the length of the MAD message, and the second long word, IU Data Pointer, is the DMA mapped address of the MAD message being passed. MAD data structures are defined later in this appendix. For MAD messages, the timeout value is not used.

E.9 CRQ VSCSI VIOS Message Format

VIOS messages are sent from the VIOS to the clients, usually in response to a request from the client. The VIOS message format is shown Table 278, “CRQ VSCSI VIOS Message,” on page 801.

Table 278. CRQ VSCSI VIOS Message

Byte Offset	0	1	2	3	4	5	6	7
0x00	CRQ Valid	CRQ Format	Reserved	Status	Reserved		IU Length	
0x08	IU TAG							

For this type of message, the first byte, CRQ Valid, must be 0x80. This same type of message is used for SRP Responses and for responses to MAD messages. If this is an SRP Response, the second byte, CRQ Format, is 0x01. If this is the response to a MAD message, the second byte is 0x02. Bytes 6 and 7 of the first long word, IU Length, contain the length of the response. The second long word contains the tag field from the original request. Both the SRP Request data structures and the MAD message data structures contain a tag field for use in this message.

The Status field of the VIOS message is for reporting special, non-SCSI status back to the client. This status is used for improving failover times in configurations where the same storage device is visible to this client over multiple adapters or when the same storage device is being shared by multiple clients in clustered configurations.

If the client enables the “fast fail” feature using the MAD_ENABLE_FAST_FAIL message, and if the VIOS determines that all paths to a device on that client adapter have failed, the VIOS will report a status of ADAPTER_FAILED (0x10) in response to a request to that device.

If the storage devices that the client are using are being shared by other clients, as is the case of an IBM General Parallel File System (GPFS™) configuration, and if the VIOS determines that all error recovery efforts on a device have failed so that there is no point in any more retries from the client, the VIOS will report a status of DEVICE_BUSY (0x08) in response to a request to that device.

In both cases (ADAPTER_FAILED and DEVICE_BUSY), the client response should be the same. The device is no longer accessible and the client should abandon any error recovery or attempts to recover access to the device using this client adapter. The client should attempt to failover to another path to the device, using another adapter, if that is possible.

E.10 Transport Events

If the first byte (CRQ Valid) of the CRQ message is 0xFF, then this message is a Transport Event from the Hypervisor and the connection to the partner has been terminated. The second byte will be the reason for the Transport Event, and may be one of the following values:

0x01 - Partner Failed. The partner partition has crashed.

0x02 - Partner de-registered the CRQ. The partner partition called H_FREE_CRQ for this CRQ. This may be as a result of error recovery, as in the case of a protocol error, or it may be the result of the system administrator removing a client or VIOS adapter.

0x06 - Client has been migrated as the result of a Partition Mobility operation. Only clients can be migrated and only clients that send the MAD_CAPABILITIES message are considered to be candidates for migration.

E.11 Messages in CRQs

If the first byte (CRQ Valid) of the CRQ message is 0x80, and the second byte (CRQ Format) is 0x06, then this is a message contained entirely within the CRQ. The rest of the message, including the IU Data Pointer, is unused and must

be initialized to zero. These messages do not require any resources on the client or VIOS, and are not subject to flow control, so may be sent at any time. However, they should be used sparingly, because they do take up an entry in the CRQ and they do require interrupt processing time to respond to them, The third byte defines the message.

Only two messages of this type have been defined to this point:

0xF5 - PING

0xF6 - PING RESPONSE

If the VIOS is not able to process interrupts, the client will likely be hung, waiting on a completion from the VIOS. To detect this condition, the client may send a PING to the VIOS. If the VIOS is capable of processing an interrupt, it responds to the PING with a PING RESPONSE, directly at interrupt level. If the client does not receive the PING RESPONSE within a reasonably short period of time, it may choose to declare the VIOS dead and attempt to failover to another client adapter. Likewise, if the VIOS for some reason needs to determine if the client is still alive, it may send a PING to the client. The client should respond as expeditiously as possible, with a PING RESPONSE.

E.12 VSCSI Management Datagrams (MADs)

VSCSI uses a number of messages that are not defined by the SRP standard. The paradigm used for these messages is the Management Datagram, discussed in the SRP and Fibre Channel specifications. Like all SRP messages, the MADs are initiated by the client partition and the VIOS responds to them. To initiate a MAD, the client sets the valid field to 0x80, sets the format field to 0x02 (MAD_FORMAT), sets the length field to the length of the data structure describing the MAD, sets the ioba field to the mapped memory address of the data structure describing the MAD, and uses the H_SEND_CRQ service provided by the Hypervisor to send the request to the VIOS.

Most of these MADs can be initiated any time after the initialization messages (INIT, INIT_COMPLETE) have been exchanged. Some of them are most appropriately done before the SRP_login message and the start of normal data transfer operations. These are: MAD_EMPTY_IU; MAD_ADAPTER_INFO_REQUEST; MAD_CAPABILITIES_EXCHANGE; and MAD_ENABLE_FAST_FAIL. Note that before the SRP_login message, resources allocated by the VIOS for a client are limited so a client should wait for one MAD to complete before issuing another, with the single exception of the MAD_EMPTY_IU message. None of them are required for normal data transfer operations between the client and VIOS. However, the MAD_ADAPTER_INFO_REQUEST provides information that customers find highly desirable, so using it is strongly recommended. In addition, the MAD_ADAPTER_INFO_REQUEST returns the size of the largest data transfer operation that the VIOS will accept from this client. Failure to honor this limit can result in client failure. And the MAD_CAPABILITIES_EXCHANGE message is required before a client is allowed to participate in partition mobility operation.

The inter_op structure is used to specify the type of MAD being sent.

```
typedef struct _inter_op_fields{
        uint32_t type;
        uint16_t status;
        uint16_t length;
        uint64_t tag;
}inter_op;
```

The type field describes the MAD and will be discussed in the paragraphs that follow.

The status field describes the result of the MAD operation. The client is required to initialize the status field to zero. The VIOS responds one of three ways:

```
#define MAD_SUCCESS 0x0
#define MAD_NOT_SUPPORTED 0xF1
#define MAD_FAILED 0xF7
```

MAD_NOT_SUPPORTED is returned if the VIOS is down-level. MAD_FAILED is returned in every other situation where the MAD did not succeed.

The length field is set to the length of the data structure(s) used in the command.

The tag field is reflected back to the client in the response to the MAD. The VIOS uses H_SEND_CRQ to send a response with the format set to 0x02 (MAD_FORMAT) and the ioba field is set to the tag field specified by the client.

The type field may be set to one of the following:

```
#define MAD_EMPTY_IU                    0x01
#define MAD_ERROR_LOGGING_REQUEST       0x02
#define MAD_ADAPTER_INFO_REQUEST        0x03
#define RESERVED                        0x04
#define MAD_CAPABILITIES_EXCHANGE       0x05
#define MAD_PHYS_ADAP_INFO_REQUEST      0x06
#define MAD_TAPE_PASSTHROUGH_REQUEST    0x07
#define MAD_ENABLE_FAST_FAIL            0x08
```

E.12.1 #define MAD_EMPTY_IU 0x01

The client sends a MAD_EMPTY_IU command if it wishes to receive an SRP target_logout before the VIOS closes the CRQ. The target_logout SRP response contains the reason that the VIOS is closing the CRQ.

The MAD_EMPTY_IU command uses the following data structure:

```
struct mad_empty_iu {
        inter_op op;
        uint64_t desp;
        uint port;
};
```

The inter_op structure is initialized with the type field set to 0x01 (MAD_EMPTY_IU), the status field set to zero, the length field set to the size of the mad_empty_iu structure, and the tag field set as described above.

The desp field is set to mapped memory address of the SRP_T_LOGOUT response data structure. The client must not unmap, free, or re-use this memory until it receives the SRP target_logout or the CRQ is closed.

The port field is unused at this time.

E.12.2 #define MAD_ERROR_LOGGING_REQUEST 0x02

The client sends the MAD_ERROR_LOGGING_REQUEST when it wishes the VIOS to write an entry in the system error log on its behalf. Hardware errors in physical storage components on the VIOS usually result in errors on the client partition using that physical storage. The MAD_ERROR_LOGGING REQUEST places client errors in the system error log in proximity to the original hardware error to enable service personnel to assess the impact of the original hardware error.

The MAD_ERROR_LOGGING_REQUEST uses the following data structure:

```
struct mad_error_logging_request{
        inter_op op;
        uint64_t buffer;
};
```

The inter_op structure is initialized with the type field set to 0x02 (MAD_ERROR_LOGGING_REQUEST), the status field set to zero, the length field set to the size of the mad_error_log structure plus the size of the buffer of additional data, if any, and the tag field set as described above.

The buffer field points to a mad_error_log structure.

```
struct mad_error_log{
        uint64_t lun;           // logical unit address
        uint64_t correlator;    // logged on both client and server in order to be
                                // able to associate an entry on the client with
                                // one on the server
        uint64_t reserved;      // future expansion
        uint32_t error_id;      // client partition specific (-1 if none is available)
        int32_t buffer_size;
                                // size of character buffer to log
        char client_name[32];   // for example “vscsi0”
        char device_name[32];   // for example “hdisk0”
        int32_t partition;      // partition number
#define LOG_DATA_BINARY 1
#define LOG_DATA_ASCII 2
        int32_t flags;          // type of data in buffer
        char buffer[1];         // start of the buffer, buffer_size bytes};
```

The lun field is set to the Logical Unit Number (LUN) of the device on the client that is logging the error.

The correlator field is optional. If used, it should have a unique value that can be used to correlate the error message on the client with the error message on the VIOS.

The error_id field is set to a client-specific number associated with the error.

The buffer_size is set to the length of the buffer of additional data, which is optional.

The client_name array is set to the name by which this client adapter instance is known on the client partition, for example “vscsi0”.

The device_name array is set to the name by which the device logging the error is known on the client partition, for example “hdisk0”.

The partition field is set to the number of the client partition requesting that the error be logged.

The flags field specifies the type of data contained in the optional buffer.

The buffer, if used, starts immediately after the mad_error_log structure. The buffer is not logged by the VIOS at this time.

E.12.3 #define MAD_ADAPTER_INFO_REQUEST 0x03

The client sends the MAD_ADAPTER_INFO_REQUEST to the VIOS to inform the VIOS of the client’s identity. The VIOS responds with the equivalent information about itself. The VIOS uses the client information provided in the MAD_ADAPTER_INFO_REQUEST for the display in the “lsmap” command. Use of this MAD is not enforced by VIOS. However, customers have found the information useful enough to insist that it be used. The MAD_ADAPTER_INFO_REQUEST may also be used after a Partition Mobility operation to allow the client to update the information on the VIOS, which may have changed during the migration.

The MAD_ADAPTER_INFO_REQUEST uses the following data structure:

```
struct mad_adapter_information_request{
        inter_op op;
        uint64_t buffer;
};
```

The inter_op structure is initialized with the type field set to 0x03 (MAD_ADAPTER_INFO_REQUEST), the status field set to zero, the length field set to the size of the mad_adapter_information_payload structure, and the tag field set as described above. The buffer field points to mapped memory address of a mad_adapter_information_payload structure.

```
typedef struct mad_adapter_information_payload{
        char srp_version[8];            // initially 16.a
        char partition_name[96];        // root node property ibm,partition-name
        uint32_t partition_number;      // root node property ibm,partition-no
#define MAD_VERSION_1 1
        uint32_t mad_version;           // initially 1
#define OS400 0x01
#define LINUX 0x02
#define AIX 0x03
#define OFW 0x04
        uint32_t os_type;
        uint32_t port_max_txu[8];
}partner_info;
```

The srp_version field is a NULL-terminated character array with the version number of the SRP standard to which the partition complies. Current versions of the VIOS and clients all support SRP revision 16.a. The VIOS does not validate or enforce this field currently.

The partition name is the ASCII string representing the name of the partition from the root node in the Open Firmware device tree.

The partition number is the integer number identifying the partition from the root node in the Open Firmware device tree. Note that partition number 0 is reserved for the hypervisor.

The mad_version field is set to the version of MAD messages supported by the partition. The MAD messages described in this document is version 1. The VIOS does not currently validate or enforce this version.

The os_type field is set to the type of Operating System being run on the partition. The VIOS uses this information to allocate additional resources for client partitions that have unique requirements and to return different values for sense data in error situations. The VIOS has been able to make minor behavior changes to the device on behalf of clients that use this field.

The port_max_txu array is used by the VIOS to report the size of the largest single request that it can handle. Currently only the first entry (port_max_txu[0]) is used. The client initializes this field to zero. The VIOS responds with at least a value of 0x40000, meaning that it is prepared to deal with a request to transfer at least 256,000 bytes of data. The VIOS can respond with a larger value, depending on the resources available and the capabilities of the physical device providing storage.

NOTE: If the VIOS reports a maximum transfer value larger than the minimum of 0x40000, and subsequently a device which cannot support that larger maximum transfer value is added to the device inventory of this host adapter, the VIOS will log an informative error and not report that new device in a REPORT_LUNS request until the client has issued another MAD adapter information request. This prevents the client from passing a data transfer request

to a device which is too large for that device to handle. The VIOS will return such requests with an error. Optical devices typically have minimal maximum transfer values.

E.12.4 #define MAD_CAPABILITIES_EXCHANGE 0x05

The MAD_CAPABILITIES_EXCHANGE command is used to allow the client and VIOS to negotiate support for capabilities that may be required with a partition migration. The data structures used are the capabilities structure, followed by at least one specific capability structure. The client uses a bit-mask to advertise the capabilities that it can support by setting the bits representing those capabilities to one. The VIOS responds by turning off (setting to zero) the bits for any capabilities that it cannot support. This allows clients and VIOSs at a variety of levels to cooperate in the partition migration operation. The client is required to support a minimum level of capabilities in order to be considered to be a candidate for migration.

The MAD_CAPABILITIES_EXCHANGE command uses the following data structure:

```
struct capabilities_mad{
        inter_op op;
        uint64_t buffer;
};
```

The inter_op field is initialized with the type field set to 0x05 (MAD_CAPABILITIES_EXCHANGE), the status field initialized to zero, the length field set to the size of the capabilities structures being passed, and the tag field set as described above. The capabilities structures must include at least the capabilities structure and the mig_cap structure.

The buffer field contains the mapped memory address of a buffer containing these structures.

```
struct capabilities{
        //Allows the server to put a LUN in the proper state
        //after migration. The flags are needed if one or
        //LUN are using client reserve
#define CLIENT_MIGRATED 0x01
#define CLIENT_RECONNECT 0x02
        //The the client should always set this flag field, it will
        //will be reset if the server found some capabilities in the
        //list it is not capable of supporting. If the server resets this
        //flag field there is at least one capability in the list it does
        //support
#define CAP_LIST_SUPPORTED 0x04
        //The server sets this flag it overwrites some filed in
        //the capabilities list. It is not set for overwriting
        //the name or location field
#define CAP_LIST_DATA 0x08
        unsigned int flags;
        //Either a Null string or NULL terminated ASCII strings.
        //If string is not NULL it may be displayed by the server
        //for the system administrator.
        char name[32];
        char loc[32];
        // list of capabilities follow
};
```

The flags field is always set to at least CAP_LIST_SUPPORTED by the client. If the client is sending this command as the result of a successful partition migration operation, it should also set the CLIENT_MIGRATED flag. If the client is sending this command as the result of a VIOS reboot or the VIOS has reset the CRQ, it should also set the

CLIENT_RECONNECT flag. If the VIOS cannot support all of the capabilities in the list passed by the client, it will turn off the CAP_LIST_SUPPORTED flag. If the VIOS overwrites some of the data in the capabilities list, it will set the CAP_LIST_DATA flag.

The name array is filled with the NULL-terminated string representing the name by which this client adapter instance is known on the client partition, for example “vscsi0”.

The loc array is filled with the NULL-terminated string from the “loc-code” field of the adapter node in the Open Firmware device tree for this client adapter, for example “U9117.MMA.107086C-V6-C5-T1”.

Following the capabilities structure is a list of capabilities to be negotiated. Capabilities currently supported by the VIOS are MIGRATION_CAPABILITIES and RESERVATION_CAPABILITIES.

```
struct capability_common{
        //Which capability
#define MIGRATION_CAPABILITIES 0x01
#define RESERVATION_CAPABILITIES 0x02
        unsigned int cap_type;
        //length of this capability
        //including the size of this structure
        //in bytes
        int16_t length;
        //client initializes to 0x01, server zeros
        //if this particular capability is not supported
#define SERVER_DOES_NOT_SUPPORTS_CAP 0x0
#define SERVER_SUPPORTS_CAP 0x01
#define SERVER_CAP_DATA 0x02
        uint16_t server_support;
};
```

The capability_common structure is included in each capability structure and describes the type of capability being negotiated.

The cap_type field is set to the type of capability. MIGRATION_CAPABILITIES and RESERVATION_CAPABILITIES are the only types of capabilities currently supported.

The length field is set to the size of the capabilities structure, currently either mig_cap or reserve_cap.

The server_support field is initialized by the client to 1. If the VIOS does not support that capability, it clears the field.

The capabilities structure used for negotiating migration capabilities is as follows:

```
struct mig_cap{
        struct capability_common common;
        unsigned int ecl;
};
```

The ecl field contains the effective capability level. The client sets it to the current migration capability level that this client is capable of supporting. If this level is lower than the level that the VIOS can support or higher than the VIOS currently supports, the VIOS sets the server_support to SERVER_CAP_DATA, sets the ecl field to the lowest level it can support or the level currently supported, as appropriate, and sets flags field of the capabilities structure to CAP_LIST_DATA, to inform the client of the difference in the levels of migration capabilities supported. Currently, the only migration capability level supported is 1.

The structure used in negotiating reservation capabilities is as follows:

```
struct reserve_cap{
        struct capability_common common;
        //allow for future expansion of different
        //types of reserves.
#define CLIENT_RESERVE_SCSI_2 0x01
        unsigned int type;
};
```

If the client is capable of breaking and re-establishing SCSI-2 reservations after a migration event, it should set the type field to CLIENT_RESERVE_SCSI_2. Otherwise, it should initialize the type field to zero.

E.12.5 #define MAD_PHYS_ADAP_INFO_REQUEST 0x06

The MAD_PHYS_ADAP_INFO_REQUEST returns data about the physical adapter to which the target device is attached, if the device supports it. The only device currently supporting this request is virtual tape. The data structure used with the MAD_PHYS_ADAP_INFO_REQUEST is as follows:

```
struct mad_phys_adapter_info_request{
        inter_op op;
        uint64_t buffer;
};
```

The client initializes the inter_op field, with the type set to 0x06 (MAD_PHYS_ADAP_INFO_REQUEST), the status field initialized to zero, the length field set to the size of the mad_phys_adapter_info structure, the tag field set as described above, and the buffer field set to the mapped memory address of a mad_phys_adapter_info structure.

```
struct mad_phys_adapter_info{
        uint64_t lun;
#define MAD_PHYS_ADAP_INFO_VERSION 0x00000001
        uint32_t version;
#ifndef MAX_FRUPN_SIZE
#define MAX_FRUPN_SIZE          128
#endif
#ifndef MAX_FRUSN_SIZE
#define MAX_FRUSN_SIZE          128
#endif
#ifndef MAX_PHYSLOC_SIZE
#define MAX_PHYSLOC_SIZE        256
#endif
        char fruPartNumber      [MAX_FRUPN_SIZE];
        char fruSerialNumber    [MAX_FRUSN_SIZE];
        char physLocationCode   [MAX_PHYSLOC_SIZE];
        char reserved           [4];
};
```

The client sets the lun field to the Logical Unit Number (LUN) of the virtual device for which it is requesting the physical adapter information, and it sets the version to 0x01 (MAD_PHYS_ADAP_INFO_VERSION).

If the target device supports returning the physical adapter information, the VIOS copies the Field Replaceable Unit (FRU) part number, the FRU serial number, and the physical location code into the appropriate arrays and returns that information to the client. This information is intended for use by customer service engineers, to assist them in repairing physical tape devices.

E.12.6 #define MAD_TAPE_PASSTHROUGH_REQUEST 0x07

The MAD_TAPE_PASSTHROUGH_REQUEST enables or disables SCSI command data blocks (CDBs) to be passed directly to the physical tape device driver without examination or emulation by the VIOS drivers.

The structure used with the MAD_TAPE_PASSTHROUGH_REQUEST is as follows:

```
struct mad_tape_passthrough{
        inter_op op;
        uint64_t lun;
#define MAD_TAPE_PASSTHRU_VERSION 0x00000001
        uint32_t version;
/*****************************************
* The below defines are used to enable or
* disable the passthrough mode for virtual
* tape devices supported by the server
*****************************************/
#define TAPE_PASSTHROUGH_ENABLE 0x00000001
#define TAPE_PASSTHROUGH_DISABLE 0x00000002
        uint32_t passThru;
};
```

The client initializes the inter_op structure by setting the type field to 0x07 (MAD_TAPE_PASSTHROUGH_REQUEST), setting the status field to zero, setting the length field to the size of the mad_tape_passthrough structure, and setting the tag field as described above. The lun field is set to the Logical Unit Number of a virtual tape device on this client adapter. The version is set to 0x00000001 (MAD_TAPE_PASSTHRU_VERSION). The passThru is set to either 0x00000001 (TAPE_PASSTHROUGH_ENABLE) or 0x00000002 (TAPE_PASSTHROUGH_DISABLE).

When tape passthrough is enabled, the SCSI Command Data Blocks are sent directly to the tape head driver, without examination or emulation by the VIOS drivers.

E.12.7 #define MAD_ENABLE_FAST_FAIL 0x08

The MAD_ENABLE_FAST_FAIL command enables the VIOS to provide a hint to the client that a physical device is no longer accessible so that a failover to alternate paths, if any, should be attempted.

The only structure used with the MAD_ENABLE_FAST_FAIL command is the inter_op structure. The type field is set to 0x08 (MAD_ENABLE_FAST_FAIL), the status field is initialized to zero, the length field is set to the size of the inter_op structure, and the tag field is set as described above.

When the MAD_ENABLE_FAST_FAIL has completed successfully and the VIOS determines that a device is no longer responding, when the VIOS is completing an I/O request for that device back to the client, the VIOS will set the status field in the CRQ message to 0x10 (ADAPTER_FAILED), in addition to returning the normal device error and sense data. Fast fail is disabled by closing the CRQ.

Two additional messages may be exchanged between clients and a VIOS - PING and PING_RESPONSE. If a partition needs to know if the other partition is still functional and at least able to respond to an interrupt, it can send a PING message to the other partition. The other partition should respond with a PING_RESPONSE. These are very lightweight messages that require no resources. They fit entirely within the first 64-bit quantity of the CRQ message. The PING_RESPONSE should be sent from the interrupt code, immediately after receiving the PING.

To send a PING, the valid bit is set to one, the CRQ format field is set to 0x06 (MESSAGE_IN_CRQ), and the status field is set to 0xF5 (PING).

To send a PING_RESPONSE, the valid bit is set to one, the CRQ format field is set to 0x06 (MESSAGE_IN_CRQ), and the status field is set to 0xF6 (PING_RESPONSE).

It is strongly recommended that PING messages be used very sparingly. One way to fill a CRQ with ping messages is to halt the VIOS in kdb while the AIX client has requests active on it.

F A Protocol for VMC Communications

F.1 Overview

The *Virtual Management Channel* (VMC) is a logical device which provides an interface between the hypervisor and a management partition. This management partition is intended to provide an alternative to HMC-based system management. In the management partition, a *Logical Partition Manager* (LPM) application exists which enables a system administrator to configure the system's partitioning characteristics via a command line interface or web browser. Support for conventional HMC management of the system may still be provided on a system; however, when an HMC is attached to the system, the VMC interface is disabled by the hypervisor.

F.1.1 Logical Partition Manager

The LPM is a browser based LPAR configuration tool provided by the management partition. System configuration, maintenance, and control functions which traditionally require an HMC can be implemented in the LPM using a combination of HMC to hypervisor interfaces and existing operating system methods. This tool provides a subset of the functions implemented by the HMC and enables basic partition configuration. The set of HMC to hypervisor messages supported by the LPM component are passed to the hypervisor over a VMC interface, which is defined below. The actual content of these messages is defined in other documentation. In order to remain consistent with this existing HMC documentation, this appendix generally uses the HMC terminology to refer to these messages and the LPM to hypervisor connections.

F.1.2 Virtual Management Channel (VMC)

A logical device, called the virtual management channel (VMC), is defined for communicating between the LPM application and the hypervisor. This device, similar to a VSCSI server device, is presented to a designated management partition as a virtual device and is only presented when the system is not HMC managed.

This communication device borrows aspects from both VSCSI and ILLAN devices and is implemented using the CRQ and the RDMA interfaces. The initialization process for CRQs is defined in Chapter 17, "Virtualized Input/Output," on page 597, and is not duplicated here. A three way handshake is defined that must take place to establish that both the hypervisor and management partition sides of the channel are running prior to sending/receiving any of the protocol messages defined in this appendix.

Transport Event CRQs are also defined in Chapter 17, "Virtualized Input/Output," on page 597, and are not duplicated here. They define the CRQ messages that are sent when the hypervisor detects one of the peer partitions has abnormally terminated, or one side has called H_FREE_CRQ to close their CRQ.

Two new classes of CRQ messages are introduced for the VMC device. *VMC Administrative* messages are used for each partition using the VMC to communicate capabilities to their partner. *HMC Interface* messages are used for the actual flow of HMC messages between the management partition and the hypervisor. As most HMC messages far exceed the size of a CRQ bugger, a virtual DMA (RMDA) of the HMC message data is done prior to each HMC Interface CRQ message. Only the management partition drives RDMA operations; hypervisor never directly causes the movement of message data.

F.2 VMC CRQ Message Definition

For the VMC interface, all CRQ messages are defined to use the following base format:

Table 279. CRQ Message Base Format

1 B	1 B	14 B
Header	Type	Data

Two general message formats are defined: *administrative* and *HMC Interface*. These are defined in the following sections.

F.2.1 Administrative Messages

The administrative message format is used to configure a VMC between the hypervisor and the management partition. The two messages defined for this format are described in the following subsections.

F.2.1.1 VMC Capabilities

Table 280. VMC Capabilities Message

1 B	1 B	1 B	2 B	1 B	2 B	4 B	2 B	2 B
0x80	0x01	Reserved	Reserved	# HMC's	Pool Size	MTU	CRQ Size	Version (Major/Minor)

The *capabilities* message is an administrative message sent after the CRQ initialization sequence of messages and is used to exchange VMC capabilities between the management partition and the hypervisor. The management partition must send this message and the hypervisor must respond with a VMC C*apabilities Response* message before HMC interface messages can begin. Any HMC interface messages received before the exchange of capabilities has completed are dropped.

This message enables the management partition and the hypervisor to trade the following interface parameters:

1. # HMC's. Maximum number of independent HMC connections supported. Multiple connections would be required to support HMC pass through mode.
2. Pool Size. Maximum number of buffers supported per HMC connection.
3. MTU. Maximum message size supported (bytes).
4. CRQ Size. Number of entries available in the CRQ for the source partition. The target partition must limit the number of outstanding messages to one half or less.
5. Version. Indicates the code level of the management partition or the hypervisor with the high-order byte indicating a major version and the low-order byte indicating a minor version.

F.2.1.2 VMC Capabilities Response

Table 281. VMC Capabilities Response Message

1 B	1 B	1 B	2 B	1 B	2 B	4 B	2 B	2 B
0x80	0x81	Status	Reserved	# HMC's	Pool Size	MTU	CRQ Size	Version (Major/Minor)

This command is sent by the hypervisor in response to the VMC Capabilities message. This command enables the hypervisor to inform the management partition of the values it supports. Parameters are identical to the VMC Capabilities message, with the addition of the following field:

Status. Zero is success. On failure, one of the following is returned:
1 - General failure
2 - Invalid version

The hypervisor and the management partition use the minimum value supported by each side for the parameters negotiated with the capabilities message exchange.

F.2.2 HMC Interface Buffers

Buffers are used to transfer data between the management partition and the hypervisor. Many of the HMC Interface messages defined in following sections indicate buffers that contain data that must be transferred. Note the following:

1. All buffers exist in the hypervisor memory, and data is moved between the hypervisor and the management partition by the management partition issuing *H_COPY_RDMA*.
2. To enable the management partition to access each buffer, the hypervisor must allocate virtual TCEs as well as the actual buffer storage.
3. Each buffer is at least the minimum negotiated *MTU* bytes long.
4. Buffers are always owned by either the management partition or the hypervisor. Management partition-owned buffers are used for messages (both commands and responses) sent to the hypervisor from the management partition. The hypervisor-owned buffers are used for messages (both responses and asynchronous events) sent from the hypervisor to the management partition.
5. Each LPM interface message carrying HMC protocol (either direction) also carries a buffer, and the ownership of this buffer transfers from sender to receiver.
6. There are no CRQ responses to the CRQ messages carrying HMC protocol. The HMC protocol responses are carried in a message sent from the other direction.
7. The maximum depth of the buffer pool is the minimum value negotiated via the capabilities exchange.
8. For each of the HMC interface commands, *Buffer ID* is used to identify the transfer buffer and ranges from 0 to *the minimum negotiated pool size - 1*.
9. There is a separate buffer pool for each LPM connection, each with the negotiated number of buffers.

F.2.3 HMC Interface Messages

There are several different HMC Interface messages, as defined in the following sections. Each CRQ message has a unique HMC Interface message type, and the HMC Interface message type defines the format for the remaining 14 bytes of data.

F.2.3.1 Interface Open

Table 282. Interface Open Command Message

1 B	1 B	1 B	1 B	1B	1 B	2 B	8 B
0x80	0x02	Reserved	Reserved	HMC Sn	HMC Idx	Buffer ID	Reserved

This command is sent by the management partition as the result of a management partition device request. It causes the hypervisor to prepare a set of data buffers for the LPM connection indicated by *HMC Idx* (HMC index). A unique HMC Idx would be used if multiple management applications running concurrently were desired. Before responding to this command, the hypervisor must provide the management partition with at least one of these new buffers (see the *Add Buffer* message defined below). The *HMC Sn* (HMC Session) field is used as a session identifier for the current VMC connection. If the management partition disconnects (for example as the result of a crash in the LPM application), the next open of the VMC device will result in the next HMC Sn value in the range from 1 to 255 being used.

This message is issued after the capabilities exchange has successfully completed and the hypervisor has issued an *Add Buffer* command to create a buffer for the management partition for use in establishing an LPM connection. The management partition sends the unique 32-byte HMC ID to the hypervisor via an RDMA using the buffer established by the hypervisor.

F.2.3.2 Interface Open Response

Table 283. Interface Open Response Message

1 B	1 B	1 B	1 B	1B	1 B	2 B	8 B
0x80	0x82	Status	Reserved	HMC Sn	HMC Idx	Buffer ID	Reserved

This command is sent by the hypervisor in response to the *Interface Open* message. The status of the open command is returned in the *Status* field. Zero is success. On failure, the following is returned:

1 - General failure

When this message is received, the indicated buffer is again available for management partition use.

F.2.3.3 Interface Close

Table 284. Interface Close Message

1 B	1 B	1 B	1 B	1B	1 B	2 B	8 B
0x80	0x03	Reserved	Reserved	HMC Sn	HMC Idx	Reserved	Reserved

This command is sent by the management partition to terminate an LPM to hypervisor connection. When this command is sent, the management partition has quiesced all I/O operations to all buffers associated with this LPM connection, and has freed any storage for those buffers.

F.2.3.4 Interface Close Response

Table 285. Interface Close Response Message

1 B	1 B	1 B	1 B	1B	1 B	2 B	8 B
0x80	0x83	Status	Reserved	HMC Sn	HMC Idx	Reserved	Reserved

This command is sent by the hypervisor in response to the LPM *Interface Close* message. The status of the close command is returned in the *Status* field. Zero is success. On failure, the following is returned:

1 - General failure

F.2.3.5 Add Buffer

Table 286. Add Buffer Message

1 B	1 B	1 B	1 B	1B	1 B	2 B	4 B	4 B
0x80	0x04	Reserved	hypervisor	HMC Sn	HMC Idx	Buffer ID	Reserved	Buffer LIOBA

This message transfers a buffer from hypervisor ownership to management partition ownership. The *LIOBA* is obtained from the virtual TCE table associated with the hypervisor side of the VMC device, and points to a buffer of size *MTU* (as established in the capabilities exchange).

The hypervisor field is set to 0 if the buffer being added is to be used by the management partition for messages inbound to the hypervisor, and to 1 if the buffer being added is to be used for messages outbound from the hypervisor.

The typical flow for adding buffers:

1. A new LPM connection is opened by the management partition.
2. The hypervisor assigns new buffers for the traffic associated with that connection.
3. The hypervisor sends VMC *Add Buffer* messages to the management partition, informing it of the new buffers.
4. The hypervisor sends an HMC protocol message (to the LPM application) notifying it of the new buffers. This informs the application that it has buffers available for sending HMC commands.

F.2.3.6 Add Buffer Response

Table 287. Add Buffer Response Message

1 B	1 B	1 B	1 B	1B	1 B	2 B	8 B
0x80	0x84	Status	Reserved	HMC Sn	HMC Idx	Buffer ID	Reserved

This command is sent by the management partition to the hypervisor in response to the *Add Buffer* message. The *Status* field indicates the result of the command. Zero is success. On failure, one of the following is returned:

1 - General failure
2 - Invalid HMC Index
3 - Invalid Buffer ID
4 - HMC connection has closed

F.2.3.7 Remove Buffer

Table 288. Remove Buffer Message

1 B	1 B	1 B	1 B	1B	1 B	2 B	8 B
0x80	0x05	Reserved	Reserved	HMC Sn	HMC Idx	Reserved	Reserved

This message requests an HMC buffer to be transferred from management partition ownership to hypervisor ownership. The management partition may not be able to satisfy the request at a particular point in time if all its buffers are in use. The management partition requires a depth of at least one inbound buffer to allow LPM commands to flow to the hypervisor. It is, therefore, an interface error for the hypervisor to attempt to remove the management partition's last buffer.

The hypervisor is expected to manage buffer usage with the LPM application directly and inform the management partition when buffers may be removed. The typical flow for removing buffers:

1. The LPM application no longer needs a communication path to a particular hypervisor function. That function is closed.
2. The hypervisor and the LPM application quiesce all traffic to that function. The hypervisor requests a reduction in buffer pool size.
3. The LPM application acknowledges the reduction in buffer pool size.
4. The hypervisor sends a *Remove Buffer* message to the management partition, informing it of the reduction in buffers.
5. The management partition verifies it can remove the buffer. This is possible if buffers have been quiesced.

F.2.3.8 Remove Buffer Response

Table 289. Remove Buffer Response Message

1 B	1 B	1 B	1 B	1B	1 B	2 B	8 B
0x80	0x85	Status	Reserved	HMC Sn	HMC Idx	Buffer ID	Reserved

This command is sent by the management partition to the hypervisor in response to the *Remove Buffer* message. The *Buffer ID* field indicates which buffer the management partition selected to remove. The *Status* field indicates the result of the command. Zero is success. On failure, the following is returned:

1 - General failure
2 - Invalid HMC Index
3 - No buffer found

F.2.3.9 Signal Message

Table 290. Signal Message

1 B	1 B	1 B	1 B	1B	1 B	2 B	4 B	4 B
0x80	0x06	Reserved	Reserved	HMC Sn	HMC Idx	Buffer ID	Reserved	Msg Len

This command is sent between the management partition and the hypervisor in order to signal the arrival of an HMC protocol message. The command can be sent by both the management partition and the hypervisor. It is used for all traffic between the LPM application and the hypervisor, regardless of who initiated the communication.

There is no response to this message.

F.3 Example Management Partition VMC Driver Interface

This section provides an example for the LPM implementation where a device driver is used to interface to the VMC device. This driver consists of a new device, for example /dev/lparvmc, which provides interfaces to open, close, read, write, and perform ioctl's against the VMC device.

F.3.1 VMC Interface Initialization

The device driver is responsible for initializing the VMC when the driver is loaded. It first creates and initializes the CRQ. Next, an exchange of VMC capabilities is performed to indicate the code version and number of resources available in both the management partition and the hypervisor. Finally, the hypervisor requests that the management partition create an initial pool of VMC buffers, one buffer for each possible HMC connection, which will be used for LPM session initialization. Prior to completion of this initialization sequence, the device returns *EBUSY* to open() calls. *EIO* is returned for all open() failures.

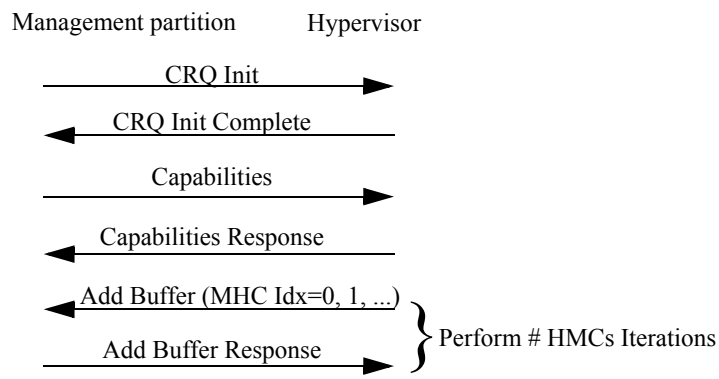


Figure 39. VMC Interface Initialization

F.3.2 VMC Interface Open

After the basic VMC channel has been initialized, an HMC session level connection can be established. The application layer performs an open() to the VMC device and executes an ioctl() against it, indicating the HMC ID (32 bytes of data) for this session. If the VMC device is in an invalid state, EIO will be returned for the ioctl(). The device driver creates a new HMC session value (ranging from 1 to 255) and HMC index value (starting at index 0 and potentially ranging to 254 in future releases) for this HMC ID. The driver then does an RDMA of the HMC ID to the hypervisor, and then sends an *Interface Open* message to the hypervisor to establish the session over the VMC. After the hypervisor receives this information, it sends *Add Buffer* messages to the management partition to seed an initial pool of buffers for the new HMC connection. Finally, the hypervisor sends an *Interface Open Response* message, to indicate that it is ready for normal runtime messaging. The following illustrates this VMC flow:

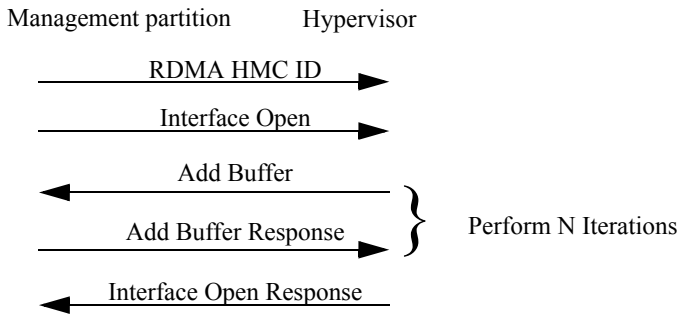


Figure 40. VMC Interface Open

F.3.3 VMC Interface Runtime

During normal runtime, the LPM application and the hypervisor exchange HMC messages via the *Signal* VMC message and RDMA operations. When sending data to the hypervisor, the LPM application performs a write() to the VMC device, and the driver RDMA's the data to the hypervisor and then sends a *Signal* Message. If a write() is attempted before VMC device buffers have been made available by the hypervisor, or no buffers are currently available, *EBUSY* is returned in response to the write(). A write() will return EIO for all other errors, such as an invalid device state. When the hypervisor sends a message to the LPM, the data is put into a VMC buffer and an *Signal Message* is sent to the VMC driver in the management partition. The driver RDMA's the buffer into the partition and passes the data up to the appropriate LPM application via a read() to the VMC device. The read() request blocks if there is no buffer available to read. The LPM application may use select() to wait for the VMC device to become ready with data to read.

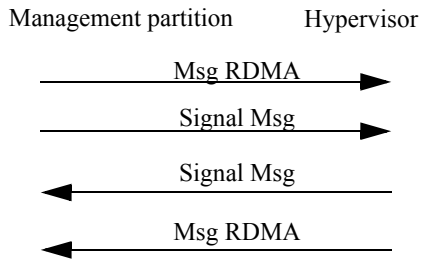


Figure 41. VMC Interface Runtime

F.3.4 VMC Interface Close

HMC session level connections are closed by the management partition when the application layer performs a close() against the device. This action results in an *Interface Close* message flowing to the hypervisor, which causes the session to be terminated. The device driver must free any storage allocated for buffers for this HMC connection.

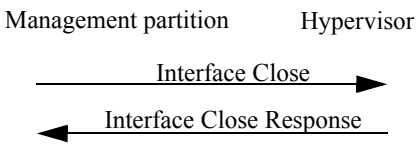


Figure 42. VMC Interface Close

G Firmware Assisted Dump Data Format

This Appendix documents the dump data format, in support of the Configure Platform Assisted Kernel Dump option (See Section 7.4.9, “ibm,configure-kernel-dump RTAS call,” on page 255).

G.1 Register Save Area

The register save area is an area in the partition’s memory used to preserve the registers for the active CPUs during a firmware assisted dump. The location and size of this area is specified by the partition when firmware assisted dump. The minimum size will be sent to the partition in the PFDS KDUMP node.

The register save is a semi-free format list of registers for each CPU. Each list of registers for a CPU starts with “CPUSTRT” and ends with “CPUEND”.

NumCpusOffset should be used to access the data to allow for additional fields to be added without affecting compatibility.

Notes:

1. Only CPUs that are online at the start of the Firmware Assisted Dump will have their register data saved.
2. Each group of GPRs, FPRs, and VRs will be listed in ascending array index (and ASCII identifier) sorted order with no other interleaving registers. Further, registers whose value spans multiple doublewords (currently only VR and VSR array elements) will be listed in high to low sorted order with no other interleaving values. All other registers are not required to be in any specific order (To make debug easier they will most likely be placed in ascending ASCII identifier order)

Table 291. Register Save Area Format

Offset	Length (Bytes)	Name	Value	Description
0x00	0x8	Magic Number	0x5245475341564500 “REGSAVE”	Identifies this area
0x08	0x4	Version	0x0	Current version
0x0C	0x4	NumCpusOffset	0x1C	Offset to NumCpus field
0x10	0xC	Padding	0x0	Must be initialized to 0
0x1C	0x4	NumCpus	Actual number of CPUs	Number of CPUs (not the number of entries)
0x20	0x10	RegEntry	“CPUSTRT”	
0x30	0x10	RegEntry	<various register values>	
...				

Table 291. Register Save Area Format ***(Continued)***

Offset	Length (Bytes)	Name	Value	Description
0x??	0x10	RegEntry	“CPUEND”	
0x??	0x10	RegEntry	“CPUSTRT”	
...				

Table 292. RegEntry Format

Offset	Length (Bytes)	Name	Value	Description
0x0	0x8	RegIdentifier	See Below	ASCII, Padded with binary zeros
0x08	0x8	RegData		Register Data

Table 293. CPUSTRT and CPUEND have the following format

8 Byte Identifier	4 Byte Reserved	4 Byte Logical CPU ID

Table 294. 8-Byte RegEntries

8 Byte Identifier	8 Byte Register Value

Table 295. 4-Byte RegEntries

8 Byte Identifier	4 Byte Reserved	4 Byte Register Value

Table 296. Identifiers Supported in Version 0x0 of the Table

Identifier (Hex)	Identifier (ASCII)	Description
0x4350555354525400	CPUSTRT	
0x435055454E440000	CPUEND	
0x41434F5000000000	ACOP	Available Coprocessor Register
0x414D520000000000	AMR	Authority Mask Register
0x4346415200000000	CFAR	Come From Address Register
0x4352000000000000	CR	Condition Register
0x4354520000000000	CTR	Count Register
0x4354524C00000000	CTRL	Control Register

Table 296. Identifiers Supported in Version 0x0 of the Table *(Continued)*

Identifier (Hex)	Identifier (ASCII)	Description
0x4441425200000000	DABR	Data Address Breakpoint Register
0x4441425258000000	DABRX	DABR Extended
0x4441520000000000	DAR	Data Address Register
0x4445430000000000	DEC	Decrementer
0x4453435200000000	DSCR	Depth Stream Control Register
0x4453495352000000	DSISR	Data Storage Interrupt Status Register
0x4650523030000000	FPR00	Floating Point Register 0
0x4650523031000000	FPR01	Floating Point Register 1
0x4650523032000000	FPR02	Floating Point Register 2
0x4650523033000000	FPR03	Floating Point Register 3
0x4650523034000000	FPR04	Floating Point Register 4
0x4650523035000000	FPR05	Floating Point Register 5
0x4650523036000000	FPR06	Floating Point Register 6
0x4650523037000000	FPR07	Floating Point Register 7
0x4650523038000000	FPR08	Floating Point Register 8
0x4650523039000000	FPR09	Floating Point Register 9
0x4650523130000000	FPR10	Floating Point Register 10
0x4650523131000000	FPR11	Floating Point Register 11
0x4650523132000000	FPR12	Floating Point Register 12
0x4650523133000000	FPR13	Floating Point Register 13
0x4650523134000000	FPR14	Floating Point Register 14
0x4650523135000000	FPR15	Floating Point Register 15
0x4650523136000000	FPR16	Floating Point Register 16
0x4650523137000000	FPR17	Floating Point Register 17
0x4650523138000000	FPR18	Floating Point Register 18
0x4650523139000000	FPR19	Floating Point Register 19
0x4650523230000000	FPR20	Floating Point Register 20
0x4650523231000000	FPR21	Floating Point Register 21
0x4650523232000000	FPR22	Floating Point Register 22
0x4650523233000000	FPR23	Floating Point Register 23
0x4650523234000000	FPR24	Floating Point Register 24
0x4650523235000000	FPR25	Floating Point Register 25

Table 296. Identifiers Supported in Version 0x0 of the Table ***(Continued)***

Identifier (Hex)	Identifier (ASCII)	Description
0x4650523236000000	FPR26	Floating Point Register 26
0x4650523237000000	FPR27	Floating Point Register 27
0x4650523238000000	FPR28	Floating Point Register 28
0x4650523239000000	FPR29	Floating Point Register 29
0x4650523330000000	FPR30	Floating Point Register 30
0x4650523331000000	FPR31	Floating Point Register 31
0x4650534352000000	FPSCR	Floating Point Status and Control Register
0x4750523030000000	GPR00	General Purpose Register 0
0x4750523031000000	GPR01	General Purpose Register 1
0x4750523032000000	GPR02	General Purpose Register 2
0x4750523033000000	GPR03	General Purpose Register 3
0x4750523034000000	GPR04	General Purpose Register 4
0x4750523035000000	GPR05	General Purpose Register 5
0x4750523036000000	GPR06	General Purpose Register 6
0x4750523037000000	GPR07	General Purpose Register 7
0x4750523038000000	GPR08	General Purpose Register 8
0x4750523039000000	GPR09	General Purpose Register 9
0x4750523130000000	GPR10	General Purpose Register 10
0x4750523131000000	GPR11	General Purpose Register 11
0x4750523132000000	GPR12	General Purpose Register 12
0x4750523133000000	GPR13	General Purpose Register 13
0x4750523134000000	GPR14	General Purpose Register 14
0x4750523135000000	GPR15	General Purpose Register 15
0x4750523136000000	GPR16	General Purpose Register 16
0x4750523137000000	GPR17	General Purpose Register 17
0x4750523138000000	GPR18	General Purpose Register 18
0x4750523139000000	GPR19	General Purpose Register 19
0x4750523230000000	GPR20	General Purpose Register 20
0x4750523231000000	GPR21	General Purpose Register 21
0x4750523232000000	GPR22	General Purpose Register 22
0x4750523233000000	GPR23	General Purpose Register 23

Table 296. Identifiers Supported in Version 0x0 of the Table *(Continued)*

Identifier (Hex)	Identifier (ASCII)	Description
0x4750523234000000	GPR24	General Purpose Register 24
0x4750523235000000	GPR25	General Purpose Register 25
0x4750523236000000	GPR26	General Purpose Register 26
0x4750523237000000	GPR27	General Purpose Register 27
0x4750523238000000	GPR28	General Purpose Register 28
0x4750523239000000	GPR29	General Purpose Register 29
0x4750523330000000	GPR30	General Purpose Register 30
0x4750523331000000	GPR31	General Purpose Register 31
0x4C52000000000000	LR	Link Register
0x4D4D435230000000	MMCR0	Monitor Mode Control Register 0
0x4D4D435231000000	MMCR1	Monitor Mode Control Register 1
0x4D4D435240000000	MMCRA	Monitor Mode Control Register A
0x4D53520000000000	MSR	Machine State Register
0x4e49410000000000	NIA	Next Instruction Address
0x5049440000000000	PID	Process ID Register
0x5049520000000000	PIR	Processor Identification Register
0x504D433100000000	PMC1	Performance Monitor Counter 1
0x504D433200000000	PMC2	Performance Monitor Counter 2
0x504D433300000000	PMC3	Performance Monitor Counter 3
0x504D433400000000	PMC4	Performance Monitor Counter 4
0x504D433500000000	PMC5	Performance Monitor Counter 5
0x504D433600000000	PIMC6	Performance Monitor Counter 6
0x5055525200000000	PURR	Processor Utilization Register
0x5056520000000000	PVR	Processor Version Register
0x5344415200000000	SDAR	Sampled Data Address Register
0x5349415200000000	SIAR	Sampled Instruction Address Register
0x5350524730000000	SPRG0	Special Purpose Register General 0
0x5350524731000000	SPRG1	Special Purpose Register General 1
0x5350524732000000	SPRG2	Special Purpose Register General 2
0x5350524733000000	SPRG3	Special Purpose Register General 3
0x5350555252000000	SPURR	Scaled Processor Utilization Register
0x5352523000000000	SRR0	Save Restore Register 0

Table 296. Identifiers Supported in Version 0x0 of the Table *(Continued)*

Identifier (Hex)	Identifier (ASCII)	Description
0x5352523100000000	SRR1	Save Restore Register 1
0x5442000000000000	TB	Time Base Register
0x5453520000000000	TSR	Thread Status Register
0x55414D4F52000000	UAMOR	User Authority Mask Override Register
0x565230305F484900	VR00_HI	Vector Register 0 High
0x565230305F4C4F00	VR00_LO	Vector Register 0 Low
0x565230315F484900	VR01_HI	Vector Register 1 High
0x565230315F4C4F00	VR01_LO	Vector Register 1 Low
0x565230325F484900	VR02_HI	Vector Register 2 High
0x565230325F4C4F00	VR02_LO	Vector Register 2 Low
0x565230335F484900	VR03_HI	Vector Register 3 High
0x565230335F4C4F00	VR03_LO	Vector Register 3 Low
0x565230345F484900	VR04_HI	Vector Register 4 High
0x565230345F4C4F00	VR04_LO	Vector Register 4 Low
0x565230355F484900	VR05_HI	Vector Register 5 High
0x565230355F4C4F00	VR05_LO	Vector Register 5 Low
0x565230365F484900	VR06_HI	Vector Register 6 High
0x565230365F4C4F00	VR06_LO	Vector Register 6 Low
0x565230375F484900	VR07_HI	Vector Register 7 High
0x565230375F4C4F00	VR07_LO	Vector Register 7 Low
0x565230385F484900	VR08_HI	Vector Register 8 High
0x565230385F4C4F00	VR08_LO	Vector Register 8 Low
0x565230395F484900	VR09_HI	Vector Register 9 High
0x565230395F4C4F00	VR09_LO	Vector Register 9 Low
0x565231305F484900	VR10_HI	Vector Register 10 High
0x565231305F4C4F00	VR10_LO	Vector Register 10 Low
0x565231315F484900	VR11_HI	Vector Register 11 High
0x565231315F4C4F00	VR11_LO	Vector Register 11 Low
0x565231325F484900	VR12_HI	Vector Register 12 High
0x565231325F4C4F00	VR12_LO	Vector Register 12 Low
0x565231335F484900	VR13_HI	Vector Register 13 High
0x565231335F4C4F00	VR13_LO	Vector Register 13 Low

Table 296. Identifiers Supported in Version 0x0 of the Table *(Continued)*

Identifier (Hex)	Identifier (ASCII)	Description
0x565231345F484900	VR14_HI	Vector Register 14 High
0x565231345F4C4F00	VR14_LO	Vector Register 14 Low
0x565231355F484900	VR15_HI	Vector Register 15 High
0x565231355F4C4F00	VR15_LO	Vector Register 15 Low
0x565231365F484900	VR16_HI	Vector Register 16 High
0x565231365F4C4F00	VR16_LO	Vector Register 16 Low
0x565231375F484900	VR17_HI	Vector Register 17 High
0x565231375F4C4F00	VR17_LO	Vector Register 17 Low
0x565231385F484900	VR18_HI	Vector Register 18 High
0x565231385F4C4F00	VR18_LO	Vector Register 18 Low
0x565231395F484900	VR19_HI	Vector Register 19 High
0x565231395F4C4F00	VR19_LO	Vector Register 19 Low
0x565232305F484900	VR20_HI	Vector Register 20 High
0x565232305F4C4F00	VR20_LO	Vector Register 20 Low
0x565232315F484900	VR21_HI	Vector Register 21 High
0x565232315F4C4F00	VR21_LO	Vector Register 21 Low
0x565232325F484900	VR22_HI	Vector Register 22 High
0x565232325F4C4F00	VR22_LO	Vector Register 22 Low
0x565232335F484900	VR23_HI	Vector Register 23 High
0x565232335F4C4F00	VR23_LO	Vector Register 23 Low
0x565232345F484900	VR24_HI	Vector Register 24 High
0x565232345F4C4F00	VR24_LO	Vector Register 24 Low
0x565232355F484900	VR25_HI	Vector Register 25 High
0x565232355F4C4F00	VR25_LO	Vector Register 25 Low
0x565232365F484900	VR26_HI	Vector Register 26 High
0x565232365F4C4F00	VR26_LO	Vector Register 26 Low
0x565232375F484900	VR27_HI	Vector Register 27 High
0x565232375F4C4F00	VR27_LO	Vector Register 27 Low
0x565232385F484900	VR28_HI	Vector Register 28 High
0x565232385F4C4F00	VR28_LO	Vector Register 28 Low
0x565232395F484900	VR29_HI	Vector Register 29 High
0x565232395F4C4F00	VR29_LO	Vector Register 29 Low

Table 296. Identifiers Supported in Version 0x0 of the Table *(Continued)*

Identifier (Hex)	Identifier (ASCII)	Description
0x565233305F484900	VR30_HI	Vector Register 30 High
0x565233305F4C4F00	VR30_LO	Vector Register 30 Low
0x565233315F484900	VR31_HI	Vector Register 31 High
0x565233315F4C4F00	VR31_LO	Vector Register 31 Low
0x5652534156450000	VRSAVE	VR Save Register
0x5653435200000000	VSCR	VMX Status and Condition Register
0x56535230305F4849	VSR00_HI	Vector Scalar Register 0 High
0x56535230305F4C4F	VSR00_LO	Vector Scalar Register 0 Low
0x56535230315F4849	VSR01_HI	Vector Scalar Register 1 High
0x56535230315F4C4F	VSR01_LO	Vector Scalar Register 1 Low
0x56535230325F4849	VSR02_HI	Vector Scalar Register 2 High
0x56535230325F4C4F	VSR02_LO	Vector Scalar Register 2 Low
0x56535230335F4849	VSR03_HI	Vector Scalar Register 3 High
0x56535230335F4C4F	VSR03_LO	Vector Scalar Register 3 Low
0x56535230345F4849	VSR04_HI	Vector Scalar Register 4 High
0x56535230345F4C4F	VSR04_LO	Vector Scalar Register 4 Low
0x56535230355F4849	VSR05_HI	Vector Scalar Register 5 High
0x56535230355F4C4F	VSR05_LO	Vector Scalar Register 5 Low
0x56535230365F4849	VSR06_HI	Vector Scalar Register 6 High
0x56535230365F4C4F	VSR06_LO	Vector Scalar Register 6 Low
0x56535230375F4849	VSR07_HI	Vector Scalar Register 7 High
0x56535230375F4C4F	VSR07_LO	Vector Scalar Register 7 Low
0x56535230385F4849	VSR08_HI	Vector Scalar Register 8 High
0x56535230385F4C4F	VSR08_LO	Vector Scalar Register 8 Low
0x56535230395F4849	VSR09_HI	Vector Scalar Register 9 High
0x56535230395F4C4F	VSR09_LO	Vector Scalar Register 9 Low
0x56535231305F4849	VSR10_HI	Vector Scalar Register 10 High
0x56535231305F4C4F	VSR10_LO	Vector Scalar Register 10 Low
0x56535231315F4849	VSR11_HI	Vector Scalar Register 11 High
0x56535231315F4C4F	VSR11_LO	Vector Scalar Register 11 Low
0x56535231325F4849	VSR12_HI	Vector Scalar Register 12 High
0x56535231325F4C4F	VSR12_LO	Vector Scalar Register 12 Low

Table 296. Identifiers Supported in Version 0x0 of the Table *(Continued)*

Identifier (Hex)	Identifier (ASCII)	Description
0x56535231335F4849	VSR13_HI	Vector Scalar Register 13 High
0x56535231335F4C4F	VSR13_LO	Vector Scalar Register 13 Low
0x56535231345F4849	VSR14_HI	Vector Scalar Register 14 High
0x56535231345F4C4F	VSR14_LO	Vector Scalar Register 14 Low
0x56535231355F4849	VSR15_HI	Vector Scalar Register 15 High
0x56535231355F4C4F	VSR15_LO	Vector Scalar Register 15 Low
0x56535231365F4849	VSR16_HI	Vector Scalar Register 16 High
0x56535231365F4C4F	VSR16_LO	Vector Scalar Register 16 Low
0x56535231375F4849	VSR17_HI	Vector Scalar Register 17 High
0x56535231375F4C4F	VSR17_LO	Vector Scalar Register 17 Low
0x56535231385F4849	VSR18_HI	Vector Scalar Register 18 High
0x56535231385F4C4F	VSR18_LO	Vector Scalar Register 18 Low
0x56535231395F4849	VSR19_HI	Vector Scalar Register 19 High
0x56535231395F4C4F	VSR19_LO	Vector Scalar Register 19 Low
0x56535232305F4849	VSR20_HI	Vector Scalar Register 20 High
0x56535232305F4C4F	VSR20_LO	Vector Scalar Register 20 Low
0x56535232315F4849	VSR21_HI	Vector Scalar Register 21 High
0x56535232315F4C4F	VSR21_LO	Vector Scalar Register 21 Low
0x56535232325F4849	VSR22_HI	Vector Scalar Register 22 High
0x56535232325F4C4F	VSR22_LO	Vector Scalar Register 22 Low
0x56535232335F4849	VSR23_HI	Vector Scalar Register 23 High
0x56535232335F4C4F	VSR23_LO	Vector Scalar Register 23 Low
0x56535232345F4849	VSR24_HI	Vector Scalar Register 24 High
0x56535232345F4C4F	VSR24_LO	Vector Scalar Register 24 Low
0x56535232355F4849	VSR25_HI	Vector Scalar Register 25 High
0x56535232355F4C4F	VSR25_LO	Vector Scalar Register 25 Low
0x56535232365F4849	VSR26_HI	Vector Scalar Register 26 High
0x56535232365F4C4F	VSR26_LO	Vector Scalar Register 26 Low
0x56535232375F4849	VSR27_HI	Vector Scalar Register 27 High
0x56535232375F4C4F	VSR27_LO	Vector Scalar Register 27 Low
0x56535232385F4849	VSR28_HI	Vector Scalar Register 28 High
0x56535232385F4C4F	VSR28_LO	Vector Scalar Register 28 Low

Table 296. Identifiers Supported in Version 0x0 of the Table *(Continued)*

Identifier (Hex)	Identifier (ASCII)	Description
0x56535232395F4849	VSR29_HI	Vector Scalar Register 29 High
0x56535232395F4C4F	VSR29_LO	Vector Scalar Register 29 Low
0x56535233305F4849	VSR30_HI	Vector Scalar Register 30 High
0x56535233305F4C4F	VSR30_LO	Vector Scalar Register 30 Low
0x56535233315F4849	VSR31_HI	Vector Scalar Register 31 High
0x56535233315F4C4F	VSR31_LO	Vector Scalar Register 31 Low
0x56535233325F4849	VSR32_HI	Vector Scalar Register 32 High
0x56535233325F4C4F	VSR32_LO	Vector Scalar Register 32 Low
0x56535233335F4849	VSR33_HI	Vector Scalar Register 33 High
0x56535233335F4C4F	VSR33_LO	Vector Scalar Register 33 Low
0x56535233345F4849	VSR34_HI	Vector Scalar Register 34 High
0x56535233345F4C4F	VSR34_LO	Vector Scalar Register 34 Low
0x56535233355F4849	VSR35_HI	Vector Scalar Register 35 High
0x56535233355F4C4F	VSR35_LO	Vector Scalar Register 35 Low
0x56535233365F4849	VSR36_HI	Vector Scalar Register 36 High
0x56535233365F4C4F	VSR36_LO	Vector Scalar Register 36 Low
0x56535233375F4849	VSR37_HI	Vector Scalar Register 37 High
0x56535233375F4C4F	VSR37_LO	Vector Scalar Register 37 Low
0x56535233385F4849	VSR38_HI	Vector Scalar Register 38 High
0x56535233385F4C4F	VSR38_LO	Vector Scalar Register 38 Low
0x56535233395F4849	VSR39_HI	Vector Scalar Register 39 High
0x56535233395F4C4F	VSR39_LO	Vector Scalar Register 39 Low
0x56535234305F4849	VSR40_HI	Vector Scalar Register 40 High
0x56535234305F4C4F	VSR40_LO	Vector Scalar Register 40 Low
0x56535234315F4849	VSR41_HI	Vector Scalar Register 41 High
0x56535234315F4C4F	VSR41_LO	Vector Scalar Register 41 Low
0x56535234325F4849	VSR42_HI	Vector Scalar Register 42 High
0x56535234325F4C4F	VSR42_LO	Vector Scalar Register 42 Low
0x56535234335F4849	VSR43_HI	Vector Scalar Register 43 High
0x56535234335F4C4F	VSR43_LO	Vector Scalar Register 43 Low
0x56535234345F4849	VSR44_HI	Vector Scalar Register 44 High
0x56535234345F4C4F	VSR44_LO	Vector Scalar Register 44 Low

Table 296. Identifiers Supported in Version 0x0 of the Table *(Continued)*

Identifier (Hex)	Identifier (ASCII)	Description
0x56535234355F4849	VSR45_HI	Vector Scalar Register 45 High
0x56535234355F4C4F	VSR45_LO	Vector Scalar Register 45 Low
0x56535234365F4849	VSR46_HI	Vector Scalar Register 46 High
0x56535234365F4C4F	VSR46_LO	Vector Scalar Register 46 Low
0x56535234375F4849	VSR47_HI	Vector Scalar Register 47 High
0x56535234375F4C4F	VSR47_LO	Vector Scalar Register 47 Low
0x56535234385F4849	VSR48_HI	Vector Scalar Register 48 High
0x56535234385F4C4F	VSR48_LO	Vector Scalar Register 48 Low
0x56535234395F4849	VSR49_HI	Vector Scalar Register 49 High
0x56535234395F4C4F	VSR49_LO	Vector Scalar Register 49 Low
0x56535235305F4849	VSR50_HI	Vector Scalar Register 50 High
0x56535235305F4C4F	VSR50_LO	Vector Scalar Register 50 Low
0x56535235315F4849	VSR51_HI	Vector Scalar Register 51 High
0x56535235315F4C4F	VSR51_LO	Vector Scalar Register 51 Low
0x56535235325F4849	VSR52_HI	Vector Scalar Register 52 High
0x56535235325F4C4F	VSR52_LO	Vector Scalar Register 52 Low
0x56535235335F4849	VSR53_HI	Vector Scalar Register 53 High
0x56535235335F4C4F	VSR53_LO	Vector Scalar Register 53 Low
0x56535235345F4849	VSR54_HI	Vector Scalar Register 54 High
0x56535235345F4C4F	VSR54_LO	Vector Scalar Register 54 Low
0x56535235355F4849	VSR55_HI	Vector Scalar Register 55 High
0x56535235355F4C4F	VSR55_LO	Vector Scalar Register 55 Low
0x56535235365F4849	VSR56_HI	Vector Scalar Register 56 High
0x56535235365F4C4F	VSR56_LO	Vector Scalar Register 56 Low
0x56535235375F4849	VSR57_HI	Vector Scalar Register 57 High
0x56535235375F4C4F	VSR57_LO	Vector Scalar Register 57 Low
0x56535235385F4849	VSR58_HI	Vector Scalar Register 58 High
0x56535235385F4C4F	VSR58_LO	Vector Scalar Register 58 Low
0x56535235395F4849	VSR59_HI	Vector Scalar Register 59 High
0x56535235395F4C4F	VSR59_LO	Vector Scalar Register 59 Low
0x56535236305F4849	VSR60_HI	Vector Scalar Register 60 High
0x56535236305F4C4F	VSR60_LO	Vector Scalar Register 60 Low

Table 296. Identifiers Supported in Version 0x0 of the Table *(Continued)*

Identifier (Hex)	Identifier (ASCII)	Description
0x56535236315F4849	VSR61_HI	Vector Scalar Register 61 High
0x56535236315F4C4F	VSR61_LO	Vector Scalar Register 61 Low
0x56535236325F4849	VSR62_HI	Vector Scalar Register 62 High
0x56535236325F4C4F	VSR62_LO	Vector Scalar Register 62 Low
0x56535236335F4849	VSR63_HI	Vector Scalar Register 63 High
0x56535236335F4C4F	VSR63_LO	Vector Scalar Register 63 Low
0x5845520000000000	XER	Fixed-Point Exception Register

G.2 Hardware Page Table Entry Save Area

The hardware page table entry save area is an area in the partition's memory used to preserve the hardware page table entries corresponding to the VRMA entries that are used when Open Firmware is started. The location and size of this area is specified by the partition when firmware assisted dump. The minimum size is reported to the partition in the new PFDS KDUMP node.

When accessing the table the offset to NumEntries should be used to allow for different versions of the table. Newer versions of the table will be compatible with all previous versions.

Table 297. HPT Entry Save Area Format

Offset	Length (Bytes)	Name	Value	Description
0x00	0x8	Magic Number	0x4850544553415645 "HPTESAVE"	Used to verify that this area contains what it should
0x08	0x4	Version	0x0	Current version
0x0C	0x4	NumEntriesOffset	0x1C	Offset to number of entries
0x10	0x8	Padding	0x0	Must be initialized to 0
0x18	0x8	NumEntries	Actual Number Of Entries	
0x20	0x18	HptEntry1		
0x38	0x18	HptEntry2		
...				

Table 298. HPT Entry Format

Offset	Length (Bytes)	Name	Value	Description
0x0	0x8	HptEntryIndex		Index into the HPT table for the entry
0x08	0x8	Dword0		HPT Entry High
0x10	0x8	Dword1		HPT Entry Low

Note: The entries are not in any particular order. It is up to the user of the save area to sort the data.

H EEH Error Processing

This appendix describes the architectural intent for EEH error processing. This appendix does not attempt to illustrate all possible scenarios, and other implementations are possible.

H.1 General Scenarios

In general, the device driver recovery consists of issuing an *ibm,read-slot-reset-state2* call prior to doing any recovery to determine if (1) the IOA is in the MMIO Stopped and DMA Stopped state (that is, that an error has occurred which has put it into this state), and (2) whether or not the PE has been reset by the platform in the process of entering the MMIO Stopped and DMA Stopped state, and then doing one of the following:

1. Simplest approach:
   - Reset the PE
   - Reconfigure the PE
2. Most general approach (detailed more in Section H.2, “More Detail on the Most General Approach,” on page 836):
   - Release the PE for *Load*/*Store*
   - Issue *Load*/*Store* instructions to get any desired state information from the IOA
   - Call the *ibm,slot-error-detail* RTAS call to get the platform error information
   - Log the error information
   - Reset the PE
   - Reconfigure the PE
3. Most robust (no reset unless necessary):
   - Release the PE for *Load*/*Store*
   - Issue *Load*/*Store* instructions to get any desired state information from the IOA
   - Call the *ibm,slot-error-detail* RTAS call to get the platform error information
   - Log the error information
   - Device driver does IOA cleanup
   - Release the PE for DMA and restart operations (no reset)

In any scenario, after several retries of a recoverable operation, the OS may determine that further recovery efforts should cease. In such a case, calling *ibm,slot-error-detail* with *Function* 2 (Permanent Error), in addition to returning error information, marks that the PE is no longer accessible due to previous errors.

H.2 More Detail on the Most General Approach

The following gives a more detailed look at scenario #2 in Section H.1, “General Scenarios,” on page 835. This will be broken up into two groups of operations: error logging and error recovery.

These scenarios assume that:

1. The *ibm,configure-pe* RTAS call is implemented.
2. The attempts at recovery stop when Max_Retries_Exceeded is true.

H.2.1 Error Logging

1. If the device driver is going to capture internal IOA-specific information as a part of the error logging process or if the IOA controlled by the device driver requires a longer wait after reset than the normal PCI specified minimum wait time, then the device driver determines whether its IOA has been reset as a result of entering EEH Stopped State, by looking at the *PE Recovery Info* output of the *ibm,read-slot-reset-state2* RTAS call.
2. The OS or device driver insures that all MMIOs to the IOA(s) in the PE are finished.
3. If the IOA requires longer wait after reset times than the specified minimum, and the PE was reset (see step #1) as a result of the EEH event, then wait the additional necessary time before continuing.
4. The OS or device driver enables PE MMIOs by calling the *ibm,set-eeh-option* RTAS call with *Function* 2.
5. The OS or device driver calls the *ibm,configure-pe* RTAS call.
    a. If the PCI fabric does not need configuring (the PE was not reset previous to the call or was reset but was previously configured with *ibm,configure-pe*), then the call returns without doing anything, otherwise it attempts to configure the fabric up to but not including the endpoint IOA configuration registers.
    b. If an EEH event occurs as a result of probing during the *ibm,configure-pe* RTAS call that results in a reset of the PE, the PE will be returned in the PE state of 2. Software does not necessarily need to check this on return from the call. The case where this occurs is expected to be rare, and probably signals a non-transient error. In this case the software can continue on with the recovery phase of the EEH processing, and will eventually hit the same EEH event on further processing.
6. If the PE was reset (see step #1) as a result of the EEH event, then if the device driver is going to gather IOA-specific information for logging, it needs to finish the configuration of the IOA PCI configuration registers, by restoring the PCI configuration space registers of the IOA(s) in the PE (for example, BARs, Memory Space Enable, etc.).
7. If desired, the device driver gathers IOA-specific information via MMIOs, by doing MMIOs to its IOA.
8. The OS or device driver calls *ibm,slot-error-detail*. Any data captured in step #7 is passed in the call. Note that maximum amount of data will be captured in some cases only when the *ibm,slot-error-detail* call is made with PE not in the MMIO Stopped State (as it should be in step #4).
    a. If Max_Retries_Exceeded is true, then call *ibm,slot-error-detail* with *Function* 2 (Permanent Error).
    b. If Max_Retries_Exceeded is not true, then call *ibm,slot-error-detail* with *Function* 1(Temporary Error).
9. The *ibm,slot-error-detail* RTAS call captures whatever PCI config space registers it can between the configuration address passed in the call and the system (PHB), and including at the configuration address and at the PHB, and returns them along with the device specific data in an error log in the return information from the call. This call may encounter another EEH event, in which case it returns what information it can in the call, with a *Status* of 0 (Success).

10. The OS or device driver logs the log entry returned from the *ibm,slot-error-detail* RTAS call.

11. If Max_Retries_Exceeded is not true, then the next step is PE Recovery, otherwise stop and mark the IOA(s) in the PE as unusable.

H.2.2 PE Recovery

1. OS or device driver does a PE reset sequence. Note that this step is required even if the PE was reset as a result of the initial EEH event, because the error logging steps (for example, the *ibm,configure-pe* or *ibm,slot-error-detail* calls) could have encountered another EEH event.

   a. The device driver or OS calls *ibm,set-slot-reset* with *Function* 1 or 3 to activate the reset.

   b. The minimum reset active time is waited.

   c. The device driver or OS calls *ibm,set-slot-reset* with *Function* 0 to deactivate the reset.

   d. The minimum reset inactive to first configuration cycles is waited. If the IOA requires more than the standard PCI specified time, then wait that longer time, instead.

2. The OS or device driver calls *ibm,configure-pe.*

   ■ **Note:** If an EEH event occurs as a result of probing during the *ibm,configure-pe* RTAS call that results in a reset of the PE, the PE will be returned in the PE state of 2. Software does not necessarily need to check this on return from the call. The case where this occurs is expected to be rare, and probably signals a non-transient error. In this case the software can continue on with the recovery phase of the EEH processing, and will eventually hit the same EEH event on further processing.

3. The device driver restores the PCI configuration spaces of the IOA(s) in the PE.

4. The device driver initializes the IOA for operations.

CMO Characteristics Definitions

This appendix defines the string that is returned by the *ibm,get-system-parameter* RTAS call when the parameter token value of 44 (CMO Characteristics) is specified on the *ibm,get-system-parameter* RTAS call as per Section 7.3.16.1, “ibm,get-system-parameter,” on page 211.

I.1 CMO Terms

The LoPAPR Cooperative Memory Over-commitment option (CMO) defines terms as presented in Table 299, “CMO Terms,” on page 839.

Table 299. CMO Terms

Term	Definition
CMO Page Size	Page size as determined by the hypervisor. CMO page size is expressed as the power of 2 of the page size. For example, a 4K page size is represented by the value of 12 ($4K = 2^{12}$).
Primary Paging Service Partition	The primary paging service partition identifies the primary VIOS which provides access to paging services and devices for partitions participating in CMO. The primary paging service partition value is the partition number of the VIOS.
Secondary Paging Service Partition	The secondary paging service partition identifies the secondary VIOS in a redundant Paging Service Partition configuration. If the hypervisor detects a problem with the primary VIOS, it fails over to the secondary VIOS. The secondary paging service partition value is the partition number of the secondary VIOS.

I.2 Key Words And Values

Table 300, “CMO Characteristics,” on page 839 defines the key words and the associated legal values that will be returned in the ASCII NULL terminated string when the parameter token value of 44 (CMO characteristics) is specified on the *ibm,get-system-parameter* RTAS call. The key word and value is separated by an ASCII equal (“=”). Each key word, value pair is delimited by an ASCII comma in the string. The numerical value of the characteristic corresponding to the key word is the decimal number that corresponds to the numeric characters in the value part of the key word, value pair.

Table 300. CMO Characteristics

Characteristics	Key Word	Values	Notes
CMO Page Size	CMOPageSize	1 - 64	
Primary Paging Service Partition	PrPSP	-1 through N	Set to -1 when the partition is not in CMO mode (i.e. is a dedicated memory partition)
Secondary Paging Service Partition	SecPSP	-1 through N	

J Platform Dependent hcall()s

This appendix defines the set of hypervisor calls (hcall()s) that are platform dependent. The existence and/or implementation of the hcall() can vary between firmware releases and between hardware platforms.

J.1 hcall()s Supported by Firmware Release & Hardware Platform

Table 301, “Platform Dependent hcall()s Supported by Release and Hardware Platform,” on page 841 is a list of platform specific hcall()s, which will be described in this appendix.

Table 301. Platform Dependent hcall()s Supported by Release and Hardware Platform

Function Name/Section	Hypervisor Call Function Token	Firmware Releases Supported	Hardware Platform Supported
Reserved	0xF000-0xF07C		
H_GetPerformanceCounterInfo / J.2.1	0xF080	eFW 3.5 and later	Power 6 and later

J.2 Supported hcall()s

J.2.1 H_GetPerformanceCounterInfo (0xF080)

This call returns information about the performance of selectable performance counters maintained by the hardware or from data collected by the Hypervisor.

Syntax:

```
int64                                          /* H_Success, H_Privilege, H_Authority */
                                               /* H_Hardware, H_Not_Available */
    hcall(const uint64 H_GetPerformanceCounterInfo  /* Retrieve performance info */
    uint64 size,                               /* Size of getPerformanceCounterInfoParms  */
    getPerformanceCounterInfoParms*)           /* Requested/Response data      */
```

Parameters:

- size – size of the getPerformanceCounterInfoParms
- getPerformanceCounterInfoParms – parameter list indicating which performance counter information to retrieve. Table 302, “Performance_Counter_Info_Parms struct,” on page 842

Semantics:

- Validate the getPerformanceInfoParms is accessible, else H_Privilege.
- Validate the size and contents of getPerformanceCounterInfoParms, else H_Parameter.

- Validate information is available for the firmware level and platform, else H_Not_Available.
- Validate partition is permitted to retrieve performance information, else H_Authority.
- Copy requested performance counters into getPerformanceInfoParms and return H_Success.

Table 302. Performance_Counter_Info_Parms struct

Member Name	Member Type	IN/OUT	Description
Requested_Information	uint_32	INPUT	See Table 303, “Performance Counter Info Requested_Information Values,” on page 842
starting_index	int_32	BOTH	At input, the partition id of the first processor/partition to retrieve performance metrics. If -1 is specified for this parameter, only information for the current processor/partition is returned. At output, the actual first processor/partition id that was found.
returned_values	uint_32	OUTPUT	Number of lists of counters returned
reserved	uint_32	N/A	Alignment
reserved	uint_64[2]	N/A	Alignment
counter_value	perf_data	BOTH	Array of counters values

The possible values for Requested_Information are as shown in Table 303, “Performance Counter Info Requested_Information Values,” on page 842.

Table 303. Performance Counter Info Requested_Information Values

Name	Value	Description
Dispatch_PURR_by_processor	0x0000 0010	The value for the counter_value is a list of values per physical processor as follows: ◆ uint64 processor time (in PURR cycles) that the processor was running work on behalf of partitions since the boot of the CEC ◆ uint32 hardware processor id ◆ uint16 owning partition id (0xFFFF is shared or unowned) ◆ uint8 processor state (0x01-Not Installed, 0x02-Guarded Off, 0x03-Unlicensed, 0x04-Shared, 0x05-Borrowed, 0x06-Dedicated) ◆ uint8[1] reserved ◆ uint32 hardware chip id (a value of -1 will be returned for Not Installed processors) ◆ uint32 hardware module id ◆ uint32 primary affinity domain ◆ uint32 secondary affinity domain ◆ uint32 processor version (a value of -1 will be returned for Not Installed processors) ◆ uint16 logical processor index ◆ uint8[10] reserved

Table 303. Performance Counter Info Requested_Information Values ***(Continued)***

Name	Value	Description
Entitled_capped _uncapped_donated _idle_PURR _by_partition	0x0000 0020	The value for the counter_value is a list of uint64 values as follows: ◆ Partition id ◆ Hypervisor collected PURR cycles that the partition was entitled to consume since the boot of the CEC (or partition creation). ◆ Hypervisor collected PURR cycles that the partition consumed as capped cycles since boot of the CEC (or partition creation). For a dedicated partition, all cycles consumed will be reported as capped cycles. For shared, these are the capped (entitled) cycles consumed by the partition. ◆ Hypervisor collected PURR cycles that the partition consumed as uncapped shared partition cycles since boot of the CEC (or partition creation). ◆ Hypervisor collected PURR cycles that were donated from a dedicated partition to uncapped partitions since boot of the CEC (or partition creation). ◆ Partition collected PURR cycles that the partition considers as idle cycles. These cycles can be subtracted from the total cycles consumed to calculate the partition's view of utilization. Note that not all operating system versions will report this value.
Run_instructions _run_cycles _by_partition	0x0000 0030	The value for the counter_value is a list of uint64 values as follows: ◆ Partition id ◆ Hypervisor collected instructions completed while the run latch is set since boot of the CEC (or partition creation). Note that this value will be zero on processors versions that do not provide the ability to collect this information. ◆ Hypervisor collected cycles while the run latch is set since boot of the CEC (or partition creation). Note that this value will be zero on processors versions that do not provide the ability to collect this information.
System_performance _capabilities	0x00000040	The value for the counter_value is a list of values for the requesting partition as follows: ◆ uint8 Is partition allowed to get performance data for other partitions (boolean). ◆ uint8[15] Reserved **Note**: This request can only be issued by a partition to obtain data about itself (i.e. *starting_index* must always be -1) *H_NOT_AVAILABLE* will be returned otherwise.
Processor_bus_utilization_ABC _links	0x00000050	The value for the counter_value is a list of values per physical chip as follows: ◆ uint32 hardware chip id ◆ uint32[3] RESERVED ◆ uint64 idle cycles for A link ◆ uint64 time value (in cycles) data for A link was collected ◆ uint64 idle cycles for B link ◆ uint64 time value (in cycles) data for B link was collected ◆ uint64 idle cycles for C link ◆ uint64 time value (in cycles) data for C link was collected

Table 303. Performance Counter Info Requested_Information Values ***(Continued)***

Name	Value	Description
Processor_bus_utilization_WXYZ_links	0x00000060	The value for the counter_value is a list of values per physical chip as follows: ◆ uint32 hardware chip id ◆ uint32[3] RESERVED ◆ uint64 idle cycles for W link ◆ uint64 time value (in cycles) data for W link was collected ◆ uint64 idle cycles for X link ◆ uint64 time value (in cycles) data for X link was collected ◆ uint64 idle cycles for Y link ◆ uint64 time value (in cycles) data for Y link was collected ◆ uint64 idle cycles for Z link ◆ uint64 time value (in cycles) data for Z link was collected
Set MMCRH (LAB ONLY)	0x8000 1000	At input, counter_value is a single value with what to set the Performance Monitor Mode Control Register H to: ◆ uint64 value to set MMCRH to in all processors There will be no output for this function other than errors. **Note 1:** A passed value of (-1) will mean that collection of these values should be disabled. **Note 2:** Whenever this value is changed, the programmable counters (HPMC1 & HPMC2) will be reset in the next collection cycle.
Retrieve HPMCx (LAB ONLY)	0x8000 2000	The value for the counter_value is a list of values per physical processor as follows: ◆ uint32 hardware processor id ◆ uint32 reserved ◆ uint64 current value of MMCRH for this processor ◆ uint64 elapsed timebase value in cycles since current MMCRH was set ◆ uint64 value for HPMC1 since current MMCRH was set ◆ uint64 value for HPMC2 since current MMCRH was set ◆ uint64 value for HPMC3 since current MMCRH was set ◆ uint64 current value for HPMC3 ◆ uint64 value for HPMC4 since current MMCRH was set ◆ uint64 current value for HPMC4

K A Protocol for VNIC Communications

K.1 Introduction

The VNIC protocol defined in this appendix defines the protocol to be used with VNIC virtual IOA, as defined in Section 17.3, “Virtual Network Interface Controller (VNIC),” on page 652. VNIC provides a mechanism which minimizes the number of times data is copied within the memory of the physical system. The virtual I/O model described herein allows for either zero copy using the redirected DMA or single copy when the data is first moved to the memory space of firmware before being DMAed to the client partition.

This protocol is designed to fulfill the following requirements:

1. Fast, efficient transfer and reception of Ethernet frames
2. Exploitation of adapter multiple transmit and receive queue support.
3. Partition mobility capable
4. Promiscuous mode support
5. Stateless TCP and IP checksum offload
6. TCP large send offload
7. Multiple interrupt source support
8. Notification of physical Ethernet link state
9. Physical Ethernet link state control if configured
10. Statistics, trace, and dump support
11. Extensible protocol for future functional additions

K.2 VNIC Adapter

The intent of this protocol is to support the implementation, within the client logical partition, of a VNIC adapter device driver (VNIC client) which is functionally similar to a physical Ethernet adapter device driver. The VNIC can send and receive Ethernet packets, add receive buffers to the virtualized hardware, handle physical and logical link status, acquire hardware statistics, and utilize advanced hardware features like checksum offload. The VNIC interface also provides tracing, logging, and dumping facilities.

It is the firm intent of this protocol that no changes be required in any layer 3 or higher communication protocol (e.g. TCP, IP, etc.).

A partition may have multiple VNIC Adapters.

K.3 Zero Copy DMA Models

Unlike the Interpartition Logical LAN option (See Chapter 17, “Virtualized Input/Output,” on page 597), the VNIC protocol allows for the physical Ethernet adapter associated with the VNIC device to securely target memory pages associated with a VNIC adapter for virtual I/O operations. LoPAPR defines two modes of LRDMA (See Chapter 17, “Virtualized Input/Output,” on page 597).

The use of Redirected RDMA is completely invisible to the VNIC adapter, and has no impact on the VNIC protocol defined here. It is left entirely to the discretion of the server firmware whether it first moves data from a physical adapter into its own memory before moving (DMAing) the data to the VNIC adapter, or whether the physical adapter sets up the I/O request in such a way that the physical device DMAs directly to the memory of the client adapter. The virtualizing firmware uses the RDMA mode that best suits its needs for a given virtual I/O operation.

K.4 Protocol Overview

The CRQ and Sub-CRQ facilities as defined in Chapter 17, “Virtualized Input/Output,” on page 597 are used to send and receive VNIC commands to system firmware. The different VNIC command types are defined in Table 306, “VNIC Command Types,” on page 847.

Throughout this document, boolean values assume 0 to be false, 1 to be true. Unless otherwise specified, all lengths are expected to be given in terms of bytes. Any setting or capability changed or enabled after a successful H_REGISTER_CRQ will be cleared when H_FREE_CRQ is performed.

The format of a VNIC command is shown in Table 304, “Format of the VNIC command,” on page 846. The Command Type field of the VNIC command is defined in Table 306, “VNIC Command Types,” on page 847. The Return Code for a a VNIC command is always at offset 12 in the response, as shown in Table 304, “Format of the VNIC command,” on page 846.

All VNIC commands have VNIC command values from 0x0 to 0x7F. Each response to a VNIC command has a VNIC command value that is equal to the command with the 0x80 bit in the command turned on.

In the event firmware receives a command it doesn’t understand, a response will be returned with an UnknownCommand return code set at offset 12, and the VNIC command type set to the passed in command type with the 0x80 bit turned on.

Table 304. Format of the VNIC command

Byte Offset	0	1	2	3	4	5	6	7
0x00	0x80	Command Type	Command dependent					
0x08	Command dependent				Return Code or Command Dependent			

Table 305. VNIC Return Code

Field Name	Byte Offset	Length	Definition
Architected Return Value	0	1	This field contains a value from Table 307, “VNIC Architected Return Values,” on page 849 that contains the high level return code for the operation.

Table 305. VNIC Return Code ***(Continued)***

Field Name	Byte Offset	Length	Definition
Detailed Error Data	1	3	This field contains an unarchitected detailed error code that can be used by firmware to further classify the error returned in the architected return value.

Table 306. VNIC Command Types

Command Type	Command value	Sent by	Description	Location
VERSION_EXCHANGE	0x01	VNIC Client	Used to inform firmware of level of protocol VNIC supports	Section K.6.1, “Version Exchange,” on page 854
VERSION_EXCHANGE_RSP	0x81	Firmware	Used to inform VNIC of level of protocol firmware supports	Section K.6.1, “Version Exchange,” on page 854
QUERY_CAPABILITY	0x02	VNIC Client	Query firmware for a specific VNIC capability	Section K.6.2, “VNIC Capabilities,” on page 854
QUERY_CAPABILITY_RSP	0x82	Firmware	Response for a QUERY_CAPABILITY	Section K.6.2, “VNIC Capabilities,” on page 854
REQUEST_CAPABILITY	0x03	VNIC Client	Request firmware to start using a specific capability value	Section K.6.2, “VNIC Capabilities,” on page 854
REQUEST_CAPABILITY_RSP	0x83	Firmware	Response from firmware to a REQUEST_CAPABILITY command	Section K.6.2, “VNIC Capabilities,” on page 854
LOGIN	0x04	VNIC Client	Used to exchange Sub-CRQ information with system firmware in preparation for functional use of the virtualized adapter	Section K.6.3, “Login Support,” on page 857
LOGIN_RSP	0x84	Firmware	Response from firmware with firmware’s Sub-CRQ information in preparation for functional use.	Section K.6.3, “Login Support,” on page 857
QUERY_PHYS_PARMS	0x05	VNIC Client	Used by VNIC client to enquire about physical port parameters such as line speed, duplex setting, etc.	Section K.6.4, “Physical Port Parameters,” on page 859
QUERY_PHYS_PARMS_RSP	0x85	Firmware	A response to the QUERY_PHYS_PARMS request containing the requested information	Section K.6.4, “Physical Port Parameters,” on page 859
QUERY_PHYS_CAPABILITIES	0x06	VNIC Client	Used by VNIC client to enquire about physical port capabilities such as line speed.	Section K.6.4, “Physical Port Parameters,” on page 859
QUERY_PHYS_CAPABILITIES_RSP	0x86	Firmware	A response to the QUERY_PHYS_CAPABILITIES request containing the requested information.	Section K.6.4, “Physical Port Parameters,” on page 859
SET_PHYS_PARMS	0x07	VNIC Client	Used by the VNIC to set physical port parameters such as line speed if allowed.	Section K.6.4, “Physical Port Parameters,” on page 859
SET_PHYS_PARMS_RSP	0x87	Firmware	Response indicating status of SET_PHYS_PARMS request	Section K.6.4, “Physical Port Parameters,” on page 859
ERROR_INDICATION	0x08	Firmware	Used to indicate to either side of an error condition.	Section K.6.10, “Error Reporting Support,” on page 871

Table 306. VNIC Command Types ***(Continued)***

REQUEST_ERROR_INFO	0x09	VNIC Client	Used to request detailed error data about a previous asynchronous error condition	Section K.6.10, “Error Reporting Support,” on page 871
REQUEST_ERROR_RSP	0x89	Firmware	Used to return detailed error data in response to a request	Section K.6.10, “Error Reporting Support,” on page 871
REQUEST_DUMP_SIZE	0x0A	VNIC Client	Used to request an estimate of how much size a VNIC collected debug dump will require.	Section K.6.7, “Dump Support,” on page 864
REQUEST_DUMP_SIZE_RSP	0x8A	Firmware	Used to inform VNIC of the dump size estimate.	Section K.6.7, “Dump Support,” on page 864
REQUEST_DUMP	0x0B	VNIC Client	Used to request firmware to perform an adapter & firmware dump to assist in problem determination	Section K.6.7, “Dump Support,” on page 864
REQUEST_DUMP_RSP	0x8B	Firmware	Used to inform VNIC Client when the requested dump has been completed	Section K.6.7, “Dump Support,” on page 864
LOGICAL_LINK_STATE	0x0C	VNIC Client	Used by VNIC Client to tell firmware to start and stop packet reception	Section K.6.5, “Logical Link State,” on page 860
LOGICAL_LINK_STATE_RSP	0x8C	Firmware	Used to inform VNIC Client of the status of the LINK_STATE request	Section K.6.5, “Logical Link State,” on page 860
REQUEST_STATISTICS	0x0D	VNIC Client	Used to retrieve standard network adapter statistics (bytes/packet sent/rcvd, etc.)	Section K.6.9, “Statistics Support,” on page 869
REQUEST_STATISTICS_RSP	0x8D	Firmware	Used to inform VNIC Client when statistics were successfully collected	Section K.6.9, “Statistics Support,” on page 869
REQUEST_RAS_COMP_NUM	0x0E	VNIC Client	Used by VNIC Client to retrieve the number of independent firmware components that can have their RAS capabilities controlled in firmware associated with the VNIC	Section K.6.8, “Reliability, Availability, and Service (RAS) Support,” on page 865
REQUEST_RAS_COMP_NUM_RSP	0x8E	Firmware	Response to the REQUEST_RAS_COMP_NUM command.	Section K.6.8, “Reliability, Availability, and Service (RAS) Support,” on page 865
REQUEST_RAS_COMPS	0x0F	VNIC Client	Used by VNIC Client to retrieve the list of component ids that can have their RAS capabilities controlled in firmware for this VNIC.	Section K.6.8, “Reliability, Availability, and Service (RAS) Support,” on page 865
REQUEST_RAS_COMPS_RSP	0x8F	Firmware	Response to the REQUEST_RAS_COMPS_RSP.	Section K.6.8, “Reliability, Availability, and Service (RAS) Support,” on page 865
CONTROL_RAS	0x10	VNIC Client	Request firmware to modify RAS characteristics to allow for easier problem determination.	Section K.6.8, “Reliability, Availability, and Service (RAS) Support,” on page 865
CONTROL_RAS_RSP	0x90	Firmware	Response to the CONTROL_RAS command.	Section K.6.8, “Reliability, Availability, and Service (RAS) Support,” on page 865
COLLECT_FW_TRACE	0x11	VNIC Client	This allows the VNIC Client to collect a trace for a firmware component.	Section K.6.8, “Reliability, Availability, and Service (RAS) Support,” on page 865
COLLECT_FW_TRACE_RSP	0x91	Firmware	Inform VNIC Client the trace collection is complete	Section K.6.8, “Reliability, Availability, and Service (RAS) Support,” on page 865
LINK_STATE_INDICATION	0x12	Firmware	Inform VNIC Client of link state changes.	Section K.6.11, “Link State Change,” on page 873

Table 306. VNIC Command Types ***(Continued)***

CHANGE_MAC_ADDR	0x13	VNIC Client	Request system firmware to change the current VNIC MAC address	Section K.6.12, “Change MAC Address,” on page 874
CHANGE_MAC_ADDR_RSP	0x93	Firmware	Inform VNIC Client of MAC address change request status	Section K.6.12, “Change MAC Address,” on page 874
MULTICAST_CTRL	0x14	VNIC Client	Request system firmware to change current multicast MAC address settings	Section K.6.13, “Multicast Support,” on page 874
MULTICAST_CTRL_RSP	0x94	Firmware	Inform VNIC Client of multicast response	Section K.6.13, “Multicast Support,” on page 874
GET_VPD_SIZE	0x15	VNIC Client	Query firmware for the size of VPD	Section K.6.14, “VPD Support,” on page 875
GET_VPD_SIZE_RSP	0x95	Firmware	Return the size of VPD to VNIC client	Section K.6.14, “VPD Support,” on page 875
GET_VPD	0x16	VNIC Client	Request system firmware to return VPD associated with adapter.	Section K.6.14, “VPD Support,” on page 875
GET_VPD_RSP	0x96	Firmware	Response to GET_VPD.	Section K.6.14, “VPD Support,” on page 875
TUNE	0x17	VNIC Client	Pass debugging information to system firmware	Section K.6.16, “Debugging Support,” on page 878
TUNE_RSP	0x97	Firmware	Response to TUNE command.	Section K.6.16, “Debugging Support,” on page 878
QUERY_IP_OFFLOAD	0x18	VNIC Client	Request details about TCP, UDP, and IP offload capabilities	Section K.6.6, “TCP, UDP, and IP Offload Support,” on page 861
QUERY_IP_OFFLOAD_RSP	0x98	Firmware	Response to QUERY_IP_OFFLOAD command.	Section K.6.6, “TCP, UDP, and IP Offload Support,” on page 861
CONTROL_IP_OFFLOAD	0x19	VNIC Client	Enable and disable TCP, UDP, and IP offload capabilities	Section K.6.6, “TCP, UDP, and IP Offload Support,” on page 861
CONTROL_IP_OFFLOAD_RSP	0x99	Firmware	Response to CONTROL_IP_OFFLOAD command.	Section K.6.6, “TCP, UDP, and IP Offload Support,” on page 861
ACL_CHANGE_INDICATION	0x1A	Firmware	Inform VNIC client of dynamic changes to access controls	Section K.6.15, “Access Control Support,” on page 876
ACL_QUERY	0x1B	VNIC Client	Request information about access control limitations in place for this VNIC.	Section K.6.15, “Access Control Support,” on page 876
ACL_QUERY_RSP	0x9B	Firmware	Response to ACL_QUERY command.	Section K.6.15, “Access Control Support,” on page 876
REQUEST_DEBUG_STATS	0x1C	VNIC Client	Request unarchitected statistics block used for debugging firmware problems.	Section K.6.9, “Statistics Support,” on page 869
REQUEST_DEBUG_STATS_RSP	0x9C	Firmware	Response to REQUEST_DEBUG_STATS command.	Section K.6.9, “Statistics Support,” on page 869

Table 307. VNIC Architected Return Values

Return Code	Value	Definition
Success	0	The requested operation completed successfully.

Table 307. VNIC Architected Return Values ***(Continued)***

Return Code	Value	Definition
PartialSuccess	1	The requested operation completed partially successful. The parameters were valid, but not all resources could be obtained to completely satisfy the command. Check the specific function definition for details.
Permission	2	The request called for permissions not available.
NoMemory	3	The request failed due to insufficient memory.
Parameter	4	One or more parameters were in error in the request.
UnknownCommand	5	The specific VNIC command is unknown.
Aborted	6	The command was aborted by some other action.
InvalidState	7	The requested command is invalid at this time.
InvalidIOBA	8	An I/O bus address passed as a parameter was invalid.
InvalidLength	9	A length passed as a parameter was invalid.
UnsupportedOption	10	A reserved value or option was used on an existing command that system firmware does not support.
Reserved	11-255	These return codes are reserved.

K.5 Typical VNIC Protocol Flows

K.5.1 Boot Flow

This section gives an overview of the typical VNIC startup sequence.

1. The operating system discovers a VNIC device in the device tree.
2. The operating system instantiates the VNIC client device driver, allocates a buffer for the VNIC CRQ, which is then TCE-mapped using the VNIC's TCE table. Since the VNIC protocol is a command/response protocol, the VNIC client should allocate a CRQ buffer big enough to handle a response for every command it wishes to have outstanding concurrently with firmware with an allowance for unsolicited asynchronous error and link state change CRQ events.
3. VNIC client calls H_REG_CRQ specifying the unit address and IOBA of the CRQ page(s), and waits for either H_Success or an INITIALIZATION message as defined in Section 17.2.3.1, "Reliable Command/Response Transport Option," on page 637.
4. VNIC client sends either an INITIALIZATION_COMPLETE or an INITIALIZATION message to firmware using H_SEND_CRQ, as defined in Section 17.2.3.1, "Reliable Command/Response Transport Option," on page 637.
5. Once the INITIALIZATION and INITIALIZATION_COMPLETE messages have been exchanged, the VNIC client sends a VERSION_EXCHANGE using H_SEND_CRQ, specifying the latest version of the VNIC protocol supported by the VNIC client.
6. Firmware responds with a VERSION_EXCHANGE_RSP specifying the version it supports. Both VNIC client and firmware must support the lower of the two versions. Until and unless the VNIC client receives a

VERSION_EXCHANGE_RSP, no further VNIC commands may be sent.

7. VNIC client may now use QUERY_CAPABILITY commands to interrogate what the firmware supports currently. Multiple QUERY_CAPABILITY commands may be send in parallel, up to one for each capability being interrogated.

8. Firmware will respond with QUERY_CAPABILITY_RSP messages for each query sent.

9. Once the queries are returned, the VNIC client uses the REQUEST_CAPABILITY commands to inform the firmware of the capabilities it plans on using. Until the capability has been requested and a successful response has been received, it will not function, and commands which use the capabilities will fail.

10. Only the Capability-related commands are usable prior to sending a Login command.

11. The VNIC client determines how many Sub-CRQs to set up based on the capabilities negotiated with the server and partition configuration, and attempts to set those up by allocating memory, mapping them with TCEs, and calling H_REG_SUB_CRQ iteratively for each Sub-CRQ.

12. Once the VNIC client has successfully gotten each Sub-CRQ it needs registered (with some possibly failing due to unavailable resources), it parcels them out to specific queues (Transmit Completion and Receive Completion), and does a REQUEST_CAPABILITY for the appropriate number of each from firmware.

13. Once the VNIC client has all SubCrqs registered, he sends a LOGIN CRQ to the server, specifying each Sub-CRQ handle and purpose as defined in the LOGIN command structure, and waits for a LOGIN_RSP which includes the server's Sub-CRQ handles and purposes.

14. Once the LOGIN_RSP has been returned successfully, the VNIC client is free to utilize the Transmit Submission Sub-CRQs and Receive Buffer Add Sub-CRQs, as well as any other VNIC command.

15. Once the VNIC client is ready to receive frames (for the Logical Link State to transition to Link Up), it sends a LOGICAL_LINK_STATE command to firmware. If the VNIC client is also in control of the physical port, sending the LOGICAL_LINK_STATE command has the side effect of initiating physical port link negotiation, as appropriate.

16. The firmware will send a LOGICAL_LINK_STATE_RSP once the link state is up.

K.5.2 Adapter reboot

In the event that system firmware encounters an error, needs to update the firmware on the adapter, or needs to remove the virtualized adapter from the partition, the following flows will happen.

1. Firmware will close its CRQ and Sub-CRQs.

2. VNIC client receives a TRANSPORT_EVENT specifying Partner Partition Closed or receives an H_Closed return code on a H_SEND_CRQ or H_SEND_SUB_CRQ hypervisor call.

3. VNIC client closes all Sub-CRQs and CRQ using H_FREE_SUB_CRQ and H_FREE_CRQ. (Optionally, only H_FREE_CRQ could be used to close the CRQ and all Sub-CRQs.)

4. VNIC client cleans up all outstanding unacknowledged transmit frames.

5. VNIC client cleans up all receive buffers that had been given to the firmware.

6. VNIC client opens the CRQ, and attempts the boot sequence.

K.5.3 Partition Mobility

In the event that an active partition is migrated to a new platform, the following sequence takes place.

1. VNIC client receives a TRANSPORT_EVENT event specifying Partner Partition Suspended (Defined in Table 232, “Transport Event Codes,” on page 622).
2. VNIC client pauses submission of new transmit frames and receive add buffers.
3. VNIC client closes all Sub-CRQs.
4. VNIC client completes all outstanding unacknowledged transmit frames. This may involve queueing them for re-transmission once the VNIC is recovered, or completing them as dropped, letting higher layers of the TCP/IP stack perform retransmission.
5. VNIC client calls H_ENABLE_CRQ until H_Success is returned.
6. VNIC client attempts the boot sequence.

K.5.4 Dump

Typical dump collection flow:

1. VNIC client decides on the need for a VNIC dump.
2. VNIC client sends a REQUEST_DUMP_SIZE command (see Table 322, “REQUEST_DUMP_SIZE and REQUEST_DUMP_SIZE_RSP Commands,” on page 864) to system firmware.
3. Firmware responds with a REQUEST_DUMP_SIZE_RSP with an estimate on the amount of storage required to store the dump into VNIC client memory.
4. VNIC client allocates a buffer big enough to hold the dump, and maps it with TCEs.
5. VNIC client sends a REQUEST_DUMP command (see Table 323, “REQUEST_DUMP Command,” on page 865) to system firmware containing the IOBAs referring to the dump buffer.
6. System firmware uses the supplied dump buffer to collect the memory that’s previously been registered by firmware as important for dumps.
7. System firmware optionally collects physical adapter debug data into the dump buffer as well.
8. System firmware sends a REQUEST_DUMP_RSP (see Table 324, “REQUEST_DUMP_RSP Command,” on page 865) to the VNIC client, indicating the dump is complete.

K.5.5 Frame Transmission

Transmission of Ethernet frames using the VNIC protocol is accomplished using two or more Subordinate CRQs. The VNIC client allocates one or more Transmit Completion Sub-CRQs and system firmware allocates one or more Transmit Submission CRQs. The handles for each are exchanged during the LOGIN processing.

The following numbered sequence details the simplified transmission of an Ethernet frame. As with any CRQ or Subordinate CRQ based protocol, the listed virtual interrupts may not occur for every CRQ or Sub-CRQ that is sent using H_SEND_CRQ, H_SEND_SUB_CRQ, or H_SEND_SUB_CRQ_INDIRECT. It is the firm intent of this protocol to allow the VNIC client and system firmware to batch frame transmission submission and transmit complete indications to minimize the number of virtual interrupts and to make the transmission of Ethernet frames as efficient as possible. Multiple Sub-CRQs may be presented to either the VNIC or system firmware with a single virtual interrupt.

1. Operating system chooses a VNIC adapter to use for frame transmission.
2. VNIC client device driver either copies the frame into a private buffer that's already been mapped via a TCE or maps the frame with a TCE.
3. VNIC client device driver constructs a Transmit Descriptor (or multiples) describing the TCE mapped buffer (see Table 352, "Transmit Descriptor Version Zero," on page 879).
4. VNIC client device driver uses H_SEND_SUB_CRQ to pass the Transmit Descriptor to system firmware's Transmit Submission Sub-CRQ.
5. System firmware receives the Sub-CRQ event, and transforms it into the appropriate format for the specific Ethernet adapter being virtualized, and uses its embedded device driver to send the frame out the wire. The system firmware uses RDMA to DMA the frame directly from the VNIC client.
6. The physical Ethernet device driver interrupts system firmware (or system firmware polls for completion at appropriate times) indicating the frame has been successfully transmitted. System firmware constructs a Transmit Completion Sub-CRQ event (see Table 353, "Transmit Completion Descriptor," on page 881), and places that Sub-CRQ onto the Transmit Completion Sub-CRQ.
7. VNIC client removes the TCE mapping for the frame, and makes it available to its network stack.

K.5.6 Frame Reception

Reception of Ethernet frames is accomplished using two or more Sub-CRQs, similar to frame transmission. System firmware creates one or more Receive Buffer Add Sub-CRQs and the VNIC client creates one or more Receive Completion Sub-CRQs.

The following numbered sequence details the simplified reception of an Ethernet frame. As with any CRQ or Subordinate CRQ based protocol, the listed virtual interrupts may not occur for every CRQ or Sub-CRQ that is sent using H_SEND_CRQ, H_SEND_SUB_CRQ, or H_SEND_SUB_CRQ_INDIRECT. It is the firm intent of this protocol to allow the VNIC client and system firmware to batch frame reception and buffer adding to minimize the number of virtual interrupts and to make the reception of Ethernet frames as efficient as possible. Multiple Sub-CRQs may be presented to either the VNIC or system firmware with a single virtual interrupt.

1. When the VNIC client is started, the VNIC allocates several memory buffers to be used to the reception of Ethernet frames. The VNIC client maps those buffers with TCEs using its TCE mapping services.
2. For each receive buffer, the VNIC client creates Add Receive Buffer Descriptor events (see Table 357, "Receive Buffer Add Descriptor," on page 884), and gives them to system firmware via the Receive Buffer Add Sub-CRQ using H_SEND_SUB_CRQ or H_SEND_SUB_CRQ_INDIRECT. Once this is done, the VNIC client should not use or otherwise modify the receive buffer until it's been given back to the VNIC client using the Receive Sub-CRQ or the Sub-CRQs and CRQ have been freed using H_FREE_SUB_CRQ and H_FREE_CRQ.
3. As system firmware receives the Receive Buffer Add Sub-CRQ events, it uses its physical adapter device driver to add the receive buffer to the physical adapter's receive queues.
4. A frame arrives for the physical adapter off of the physical wire, and the adapter dmas the frame directly to the VNIC client's memory for one of the receive buffers.
5. System firmware receives an interrupt from the physical adapter saying a frame has arrived, and uses the information it saves to generate a Receive Completion event Sub-CRQ (see Table 356, "Receive Completion Descriptor," on page 883), and places it on the appropriate Receive Completion Sub-CRQ.
6. The VNIC client receives a virtual interrupt for its Receive Completion Sub-CRQ, and passes the frame up its network stack.

K.6 VNIC Commands

All VNIC commands are sent using H_SEND_CRQ.

K.6.1 Version Exchange

The VERSION_EXCHANGE command as defined in Table 308, “VERSION_EXCHANGE and VERSION_EXCHANGE_RSP Command,” on page 854 allow the VNIC protocol to be easily updated in the future. Each side is required to support the highest common version of the VNIC protocol specification, as exchanged right after the low level CRQ registration flows.

Table 308. VERSION_EXCHANGE and VERSION_EXCHANGE_RSP Command

Field Name	Byte Offset	Length	Definition
CRQ Type	0	1	This field should be set to 0x80 to indicate a valid CRQ event.
VNIC Command	1	1	This field should be either VERSION_EXCHANGE or VERSION_EXCHANGE_RSP.
Version	2	2	Maximum version that VNIC client supports on a VERSION_EXCHANGE and the maximum version that system firmware supports on a VERSION_EXCHANGE_RSP. Each side must support the highest common version between the two versions. A value from Table 309, “VNIC Protocol Versions,” on page 854 will be contained in this field.
Reserved	4	8	This field is reserved, and should be set to 0.
Return Code	12	4	This is a return code for the operation as defined in Table 305, “VNIC Return Code,” on page 846.

Table 309. VNIC Protocol Versions

Value	Definition
1	Initial VNIC protocol version
2-65535	Reserved

K.6.2 VNIC Capabilities

The VNIC capabilities command as defined in Table 310, “CAPABILITIES Commands,” on page 855 is used to create an abstracted architecture for discovering and utilizing different NIC advanced functions on adapters, in an adapter-independent manner. As new capabilities are introduced in adapters, more capability values will be added.

To discover which capabilities a VNIC currently supports, multiple QUERY_CAPABILITY commands should be sent from the VNIC client for each capability of interest. System firmware will return the current capability setting, or a bad return code if the capability isn’t supported. System firmware will return UnsupportedOption for any capability it doesn’t understand.

If the VNIC client wishes to use one of the supported capabilities, it must be enabled via the correct REQUEST_CAPABILITY command. If a capability has a variable number of settings (settable via the Number field),

and system firmware doesn't support the value, a PartialSuccess return code will be returned with the capped value in the response.

If the VNIC client wishes to use REQUEST_CAPABILITY to determine if any specific capabilities are valid without performing QUERY_CAPABILITY commands, that's acceptable, but there may be side effects as a result.

Table 310. CAPABILITIES Commands

Field Name	Byte Offset	Length	Definition
CRQ Type	0	1	This field should be set to 0x80 to indicate a valid CRQ event.
VNIC Command	1	1	This field should be QUERY_CAPABILITIES, REQUEST_CAPABILITIES, or their associated responses.
Capability	2	2	This value should be one of the values from Table 311, "VNIC Capabilities," on page 855.
Number	4	8	This field is used for both REQUESTs and QUERYs. For a REQUEST, it's the value that the VNIC client wishes to use. On any RSP, it's the new (or unchanged) current value of the capability.
Return Code	12	4	This is a return code for the operation as defined in Table 305, "VNIC Return Code," on page 846.

Table 311. VNIC Capabilities

Value	Field Name	Behavior
1	Minimum Number of firmware-supported Transmit Completion/Submission Queues	- Query-only - Integer value returned in Number
2	Minimum Number of firmware-supported Receive Completion Queues	- Query-only - Integer value returned in Number
3	Minimum Number of firmware-supported Receive Buffer Add Queues per Receive Completion Queue	- Query-only - Integer value returned in Number
4	Maximum Number of firmware-supported Transmit Completion/Submission Queues	- Query-only - Integer value returned in Number
5	Maximum Number of firmware-supported Receive Completion Queues	- Query-only - Integer value returned in Number
6	Maximum Number of firmware-supported Receive Buffer Add Queues per Receive Completion Queue	- Query-only - Integer value returned in Number
7	Requested Number of Transmit Completion/Submission Queues	- Settable - Positive integer value set and returned in Number
8	Requested Number of Receive Completion Queues	- Settable - Positive integer value set and returned in Number
9	Requested Number of Receive Buffer Add Queues per Receive Completion Queue	- Settable - Positive integer value set and returned in Number
10	Minimum Number of Transmit Entries Per Sub-CRQ	- Query only - Positive integer value set and returned in Number

Table 311. VNIC Capabilities ***(Continued)***

Value	Field Name	Behavior
11	Minimum Number of Receive Buffer Add Entries per Sub-CRQ	- Query only - Positive integer value set and returned in Number
12	Maximum Number of Transmit Entries Per Sub-CRQ	- Query only - Positive integer value set and returned in Number
13	Maximum Number of Receive Buffer Add Entries per Sub-CRQ	- Query only - Positive integer value set and returned in Number
14	Requested Number of Transmit Entries Per Sub-CRQ	- Settable - Positive integer value set and returned in Number
15	Requested Number of Receive Buffer Add Entries per Sub-CRQ	- Settable - Positive integer value set and returned in Number
16	TCP/IP offload supported	- Query only - Boolean value returned in Number. If TRUE, TCP/IP offload commands defined in Section K.6.6, “TCP, UDP, and IP Offload Support,” on page 861 are supported.
17	Promiscuous mode requested	- Settable - Boolean value returned in Number
18	Promiscuous mode supported	- Query only - Boolean value returned in Number
19	Minimum MTU size	- Query only - Positive integer value set and returned in Number
20	Maximum MTU size	- Query only - Positive integer value set and returned in Number
21	Requested MTU size	- Settable - Positive integer value set and returned in Number - This setting can impact the minimum number of queues or receive buffer sizes supported, and should either be set early or other capabilities will need to be reevaluated.
22	Maximum Number of Unique Multicast MAC address filters	- Query only - Positive integer value set and returned in Number
23	VLAN Header insertion supported	- Query only - Boolean value returned in Number - This is controlled on a packet by packet basis in the transmit descriptor.
24	Reserved	
25	Maximum Transmit Scatter Gather entries	- Query only - Positive integer value reflecting the maximum number of IOBAs that can be used to describe a single frame using Transmit Descriptors.
26	Receive Scatter/Gather Mode supported	- Query only - Boolean value set and returned in Number. - If supported, this can enable chaining of receive buffers together to minimize the amount of memory that needs to be added in the form of Receive Buffers (Particularly if Large Receive Offload is enabled).

Table 311. VNIC Capabilities ***(Continued)***

Value	Field Name	Behavior
27	Receive Scatter/Gather Mode Requested	- Settable - Boolean value set and returned in Number. - This setting can impact the number of queues or receive buffer sizes supported, and should be set before configuring those values.
28-65535	Reserved	

K.6.3 Login Support

The use of the LOGIN and LOGIN_RSP commands is defined in Section K.5, “Typical VNIC Protocol Flows,” on page 850. The format of the LOGIN command is defined in Table 312, “LOGIN Request,” on page 857, and the LOGIN_RSP command is defined in Table 315, “LOGIN_RSP Command,” on page 859.

There must be exactly one Transmit Submission Sub-CRQ for each Transmit Completion Sub-CRQ, and vice versa. Each is implicitly tied to the other by virtue of the order each appears in the array of handles in the LOGIN Buffer and LOGIN Response buffer. (i.e. The first entry in each are associated, the second entry in each are associated, etc.)

Table 312. LOGIN Request

Field Name	Byte Offset	Length	Definition
CRQ Type	0	1	This should be set to 0x80 to indicate a valid CRQ event.
VNIC Command	1	1	This should be set to LOGIN.
Reserved	2	6	This field is reserved, and should be set to 0.
IOBA	8	4	This field is an I/O bus address referring to a TCE-mapped buffer containing the LOGIN Buffer as defined in Table 313, “LOGIN Buffer,” on page 857.
Length	12	4	This field is the length of the TCE-mapped LOGIN buffer. This value should match that as seen in the LOGIN Buffer.

Table 313. LOGIN Buffer

Field Name	Byte Offset	Length	Definition
Total Length	0	4	This field is the total length of the LOGIN Buffer.
Version	4	4	This field contains the version of LOGIN Buffer layout. The initial version should be set to 1.
Number of Transmit Completion Sub-CRQs	8	4	This field contains the number of Transmit Completion Sub-CRQs as allocated by the VNIC client.
Offset to Transmit Completion Sub-CRQ handles	12	4	Offset from the beginning of the LOGIN buffer to the start of an array of 8 byte elements containing the array of Transmit Completion Sub-CRQ handles.
Number of Receive Completion Sub-CRQs	16	4	This field contains the number of Receive Completion Sub-CRQs as allocated by the VNIC client.

Table 313. LOGIN Buffer ***(Continued)***

Field Name	Byte Offset	Length	Definition
Offset to Receive Completion Sub-CRQ handles	20	4	Offset from the beginning of the LOGIN Buffer to the start of an array of 8 byte elements containing the array of Receive Completion Sub-CRQ handles.
Login Response buffer IOBA	24	4	This field contains an I/O buffer address referencing a TCE-mapped buffer to be used for the system firmware to place its LOGIN Response buffer containing its variable length array of Sub-CRQ handles. This can point to the same storage as the LOGIN Buffer, as necessary.
Login Response buffer length	28	4	This field contains the length of the Login Response buffer described in the Login Response buffer IOBA. The VNIC client needs to ensure that system firmware will have enough space to place each Sub-CRQ handle as requested prior to LOGIN using REQUEST_CAPABILITY commands.
Transmit Completion Sub-CRQ handle array	variable	variable	This is a variable sized array containing the Sub-CRQ handles obtained from H_REGISTER_SUB_CRQ for the Transmit Completion Sub-CRQ handles.
Receive Completion Sub-CRQ handle array	variable	variable	This is a variable sized array containing the Sub-CRQ handles obtained from H_REGISTER_SUB_CRQ for the Receive Completion Sub-CRQ handles.

Table 314. LOGIN Response Buffer

Field Name	Byte Offset	Length	Definition
Total Length	0	4	This field is the total length of the Login Response Buffer.
Version	4	4	This field contains the version of LOGIN Response Buffer layout. The initial version should be set to 1.
Number of Transmit Submission Sub-CRQs	8	4	This field contains the number of Transmit Submission Sub-CRQs as requested by the VNIC client and allocated by firmware.
Offset to Transmit Submission Sub-CRQ handles	12	4	Offset from the beginning of the LOGIN Response Buffer to the start of an array of 8 byte elements containing the array of Transmit Submission Sub-CRQ handles.
Number of Receive Buffer Add Sub-CRQs	16	4	This field contains the total number of Receive Buffer Add Sub-CRQs as requested by the VNIC client and allocated by the firmware. The first n correspond to the first Receive Completion Sub-CRQ, the next n to the second, etc., where n is the Requested number of Receive Buffer Add Sub-CRQs per Receive Completion Queue requested by the VNIC client.
Offset to Receive Buffer Add Sub-CRQ handles	20	4	Offset from the beginning of the LOGIN Response Buffer to the start of an array of 8 byte elements containing the array of Receive Buffer Add Sub-CRQ handles.
Offset to Receive Buffer Add Buffer Size	24	4	Offset from the beginning of the LOGIN Response Buffer to the start of an array of 8 byte sizes. There is one size for each Receive Buffer Add Sub-CRQ, and each size represents the receive buffer size possible for that specific Receive Buffer Add Sub-CRQ.
Number of Supported Transmit Descriptors	28	4	This field contains the number of supported Transmit Descriptors, as detailed in Section K.7.1, “Frame Transmission,” on page 879.
Offset to Supported Transmit Descriptors array	32	4	Offset from the beginning of the LOGIN Response Buffer to the start of an array of 1 byte values. There is one value for each supported Transmit Descriptor format, sorted so the formats with best performance will be first in the array.

Table 314. LOGIN Response Buffer ***(Continued)***

Field Name	Byte Offset	Length	Definition
Transmit Submission Sub-CRQ handle array	variable	variable	This is a variable sized array containing the Sub-CRQ handles obtained from H_REGISTER_SUB_CRQ for the Transmit Submission Sub-CRQ handles.
Receive Buffer Add Sub-CRQ handle array	variable	variable	This is a variable sized array containing the Sub-CRQ handles obtained from H_REGISTER_SUB_CRQ for the Receive Buffer Add Sub-CRQ handles.
Receive Buffer Add Buffer Size array	variable	variable	This is a variable sized array containing the Receive Buffer Sizes must use for the respective Receive Buffer Add Sub-CRQs.
Supported Transmit Descriptors array	variable	variable	This is a variable sized array containing the performance-order sorted array of one byte supported Transmit Descriptor formats.

Table 315. LOGIN_RSP Command

Field Name	Byte Offset	Length	Definition
CRQ Type	0	1	This should be set to 0x80 to indicate a valid CRQ event.
VNIC Command	1	1	This should be set to LOGIN_RSP.
Reserved	2	10	This field is reserved, and should be set to 0.
Return Code	12	4	This is a return code for the operation as defined in Table 305, “VNIC Return Code,” on page 846.

K.6.4 Physical Port Parameters

A VNIC client may always use the QUERY_PHYS_PARM command to retrieve information about the current physical port state such as current link speed and state.

A VNIC client may always use the QUERY_PHYS_CAPABILITIES command to retrieve information about the current capabilities of the physical adapter associated with the VNIC, including allowed speed, duplex, and ability to modify those values.

If the VNIC client wishes to determine all bits that are supported by firmware, it may choose to send a QUERY_PHYS_CAPABILITIES command with no bits turned on. Firmware will respond with all possible bits it supports. If the VNIC client wishes to determine if a specific bit combination is supported by firmware, it may turn on those specific bit combinations. In that case, firmware will validate the combination, and validate the specific combination.

If the system administrator has configured the VNIC to have physical port configuration authority, the VNIC client may also use the SET_PHYS_PARMS command to change those values.

The SET_PHYS_PARMS, QUERY_PHYS_PARMS, and QUERY_PHYS_CAPABILITIES commands all use a common command format defined in Table 316, “Physical Port Parameters Commands,” on page 860.

Table 316. Physical Port Parameters Commands

Field Name	Byte Offset	Length	Definition
CRQ Type	0	1	This should be set to 0x80 to indicate a valid CRQ event.
VNIC Command	1	1	This field will be either SET_PHYS_PARMS, QUERY_PHYS_PARMS, QUERY_PHYS_CAPABILITIES, or the respective response values.
Flags	2	1	The following bits are used to either request the specific capability on a SET_PHYS_PARMS, indicate the capability to use that capability on a QUERY_PHYS_CAPABILITIES, or return the current value of the capability on a QUERY_PHYS_PARMS command. Bit 0: External loopback mode Bit 1: Internal loopback mode Bit 2: Promiscuous mode Bit 3: Physical Link Active (VNIC can communicate onto the physical media) Bit 4: Autonegotiate Duplex Bit 5: Full duplex mode Bit 6: Half duplex mode Bit 7: If set, the VNIC has the ability to change physical port parameters.
Flags	3	1	Bit 0: Adapter Logical Link Active (multiple VNICs on the same adapter can communicate) Bit 1-7: Reserved
Speed	4	4	The following bits are used to either request the specific speed on a SET_PHYS_PARMS, indicate the capability to use that speed on a QUERY_PHYS_CAPABILITIES, or return the current speed on a QUERY_PHYS_PARMS command. Bit 0: Autonegotiate speed Bit 1: 10 megabit speed Bit 2: 100 megabit speed Bit 3: 1 gigabit speed Bit 4: 10 gigabit speed Bit 5-31: These bits are reserved, and should be set to 0.
MTU	8	4	This field is used to request a MTU for this VNIC client, and to return the current MTU setting. If this value exceeds the allowed value in the case where there are multiple VNICs associated with the same physical adapter, this MTU will be capped to an allowable value, and a PartialSuccess return code will be returned on the SET_PHYS_PARMS command.
Return Code	12	4	On a response, this is a return code for the operation as defined in Table 305, “VNIC Return Code,” on page 846.

K.6.5 Logical Link State

When the VNIC does not have authority to change the physical port parameters, the LOGICAL_LINK_STATE command and response provide a method for the VNIC to inform system firmware when it’s ready to receive packets. The format of the LOGICAL_LINK_STATE and LOGICAL_LINK_STATE_RSP commands is defined in Table 317, “LOGICAL_LINK_STATE and LOGICAL_LINK_STATE_RSP commands,” on page 861.

The current VNIC logical link state will always be returned in the Link State field on a LOGICAL_LINK_STATE_RSP.

Table 317. LOGICAL_LINK_STATE and LOGICAL_LINK_STATE_RSP commands

Field Name	Byte Offset	Length	Definition
CRQ Type	0	1	This should be set to 0x80 to indicate a valid CRQ event.
VNIC Command	1	1	This field will be either LOGICAL_LINK_STATE or LOGICAL_LINK_STATE_RSP.
Link State	2	1	This field is used to request a logical link state change by the VNIC client without a corresponding change to the physical link state. The intended use for this is when a VNIC client is associated with a NIC VF that doesn't have control over the physical port to control when the VNIC client receives incoming frames. If this field is a 0, the link should be down, if the field is a 1, the link should be up. if this field is 0xFF, no logical link state change will be done, and the current logical link state will be returned in the response.
Reserved	3	9	This field is reserved, and should be set to 0.
Return Code	12	4	This is a return code for the operation as defined in Table 305, "VNIC Return Code," on page 846.

K.6.6 TCP, UDP, and IP Offload Support

The QUERY_IP_OFFLOAD command as defined in Table 318, "QUERY_IP_OFFLOAD and QUERY_IP_OFFLOAD_RSP Commands," on page 861 allows the VNIC client to determine what facilities exist in the VNIC system firmware, and its limitations, if any.

Based on the capabilities and limitations, the CONTROL_IP_OFFLOAD command as defined in Table 319, "CONTROL_IP_OFFLOAD and CONTROL_IP_OFFLOAD_RSP Command," on page 862 allows the VNIC client to enable appropriate offload capabilities. QUERY_IP_OFFLOAD and CONTROL_IP_OFFLOAD must be done prior to successful LOGIN exchange.

All offload parameters are off by default.

Table 318. QUERY_IP_OFFLOAD and QUERY_IP_OFFLOAD_RSP Commands

Field Name	Byte Offset	Length	Definition
CRQ Type	0	1	This should be set to 0x80 to indicate a valid CRQ event.
VNIC Command	1	1	This field will be either QUERY_IP_OFFLOAD or QUERY_IP_OFFLOAD_RSP.
Reserved	2	2	This field is reserved, and should be set to 0.
Length	4	4	This field contains the length of the QUERY_IP_OFFLOAD buffer
IOBA	8	4	This field is an I/O bus address referring to a TCE-mapped buffer used by system firmware to return IP offload information. On reception of a successful QUERY_IP_OFFLOAD_RSP, the buffer will be filled in with the structure as defined in Table 320, "QUERY_IP_OFFLOAD Buffer," on page 862.
Return Code	12	4	This is a return code for the operation as defined in Table 305, "VNIC Return Code," on page 846.

Table 319. CONTROL_IP_OFFLOAD and CONTROL_IP_OFFLOAD_RSP Command

Field Name	Byte Offset	Length	Definition
CRQ Type	0	1	This should be set to 0x80 to indicate a valid CRQ event.
VNIC Command	1	1	This field will be CONTROL_IP_OFFLOAD or CONTROL_IP_OFFLOAD_RSP.
Reserved	2	2	This field is reserved, and should be set to 0.
IOBA	4	4	This field is an I/O bus address referring to a TCE-mapped buffer containing the parameters to enable or disable TCP, UDP, and IP offload. The format of this buffer is defined in Table 321, “CONTROL_IP_OFFLOAD Buffer,” on page 863.
Length	8	4	This field contains the length of the CONTROL_IP_OFFLOAD buffer.
Return Code	12	4	This is a return code for the operation as defined in Table 305, “VNIC Return Code,” on page 846.

Table 320. QUERY_IP_OFFLOAD Buffer

Field Name	Byte Offset	Length	Definition
Total Length	0	4	This field is the total length of the QUERY_IP_OFFLOAD Buffer.
Version	4	4	This field contains the version of QUERY_IP_OFFLOAD Buffer layout. The initial version should be 1.
IPv4 Checksum offload supported	8	1	This field is 1 if supported
IPv6 Checksum offload supported	9	1	This field is 1 if supported.
TCP over IPv4 checksum offload supported	10	1	This field is 1 if supported.
TCP over IPv6 checksum offload supported	11	1	This field is 1 if supported.
UDP over IPv4 checksum offload supported	12	1	This field is 1 if supported.
UDP over IPv6 checksum offload supported	13	1	This field is 1 if supported.
Large send offload over IPv4 supported	14	1	This field is 1 if supported.
Large send offload over IPv6 supported	15	1	This field is 1 if supported.
Large receive offload over IPv4 supported	16	1	This field is 1 if supported.
Large receive offload over IPv4 supported	17	1	This field is 1 if supported.
Reserved	18	14	This field is reserved, and is set to 0.

Table 320. QUERY_IP_OFFLOAD Buffer ***(Continued)***

Field Name	Byte Offset	Length	Definition
Maximum IPv4 header size	32	2	This field contains the maximum size of the IPv4 header for offload operations, or 0xFFFF if no limit.
Maximum IPv6 header size	34	2	This field contains the maximum size of the IPv6 header for offload operations, or 0xFFFF if no limit.
Maximum TCP header size	36	2	This field contains the maximum size of the TCP header for offload operations, or 0xFFFF if no limit.
Maximum UDP header size	38	2	This field contains the maximum size of the UDP header for offload operations, or 0xFFFF if no limit.
Maximum Large send offload size	40	4	This field contains the maximum size of a pseudo-frame for large send offload operations, or 0xFFFFFFFF if no limit.
Maximum Large receive offload size	44	4	This field contains the maximum size of a pseudo-frame for large receive offload operations, or 0xFFFFFFFF if no limit.
Reserved	48	16	This field is reserved, and is set to 0.
IPv6 Extension Header supported	64	1	This field contains a 0 if no extension headers are supported. This field contains a 1 if extension headers are supported with limits This field contains a 0xFF if all IPv6 extension headers are supported
TCP Pseudosum required	65	1	This field is 0 if no pseudosum is required in the frame. This field is 1 if a standard pseudosum is required to be put in the frame. All other values are reserved.
Reserved	66	30	This field is reserved, and is set to 0.
Number of IPv6 extension headers supported	96	2	This field must be non zero if the IPv6 Extension Header supported field is 1.
Offset to List of supported IPv6 extension headers	98	4	This field contains an offset from the start of the QUERY_IP_OFFLOAD buffer to the array of supported extension header values.
Reserved	102	154	This field is reserved, and should be set to 0.
Array of IPv6 extension header types	variable	variable	This is an array of one byte values that are the extension header types supported by IPv6 offload.

Table 321. CONTROL_IP_OFFLOAD Buffer

Field Name	Byte Offset	Length	Definition
Total Length	0	4	This field is the total length of the CONTROL_IP_OFFLOAD Buffer.
Version	4	4	This field contains the version of CONTROL_IP_OFFLOAD Buffer layout. The initial version should be set to 1.
Enable IPv4 Checksum offload	8	1	This field is 1 if desired
Enable IPv6 Checksum offload	9	1	This field is 1 if desired.
Enable TCP over IPv4 checksum offload	10	1	This field is 1 if desired.

Table 321. CONTROL_IP_OFFLOAD Buffer ***(Continued)***

Field Name	Byte Offset	Length	Definition
Enable TCP over IPv6 checksum offload	11	1	This field is 1 if desired.
Enable UDP over IPv4 checksum offload	12	1	This field is 1 if desired.
Enable UDP over IPv6 checksum offload	13	1	This field is 1 if desired.
Enable Large send offload over IPv4	14	1	This field is 1 if desired.
Enable Large send offload over IPv6	15	1	This field is 1 if desired.
Enable bad packet reception	16	1	This field is 1 if desired
Reserved	17	111	This field is reserved, and should be set to 0.

K.6.7 Dump Support

The dumps collected via the VNIC interface is a smart dump that depends upon a working device driver. If the portion of system firmware servicing the physical adapter has run into catastrophic problems, an entire memory image of the associated firmware will be collected automatically, and collected in a similar fashion to platform dumps.

If the VNIC client detects that the VNIC interface is not providing the services in the manner it expects, it may utilize the dump support to collect focused debugging data collected and stored in VNIC client storage that's been TCE-mapped.

The format of the REQUEST_DUMP command is defined in Table 323, "REQUEST_DUMP Command," on page 865, and the format of the REQUEST_DUMP_RSP command is defined in Table 324, "REQUEST_DUMP_RSP Command," on page 865.

Table 322. REQUEST_DUMP_SIZE and REQUEST_DUMP_SIZE_RSP Commands

Field Name	Byte Offset	Length	Definition
CRQ Type	0	1	This should be set to 0x80 to indicate a valid CRQ event.
VNIC Command	1	1	This field will be either REQUEST_DUMP_SIZE or REQUEST_DUMP_SIZE_RSP.
Reserved	2	6	This field is reserved, and should be set to 0.
Length	8	4	This field is set to the estimated length of the VNIC dump in the REQUEST_DUMP_RSP if the return code was Success.
Return Code	12	4	This is a return code for the operation as defined in Table 305, "VNIC Return Code," on page 846.

Table 323. REQUEST_DUMP Command

Field Name	Byte Offset	Length	Definition
CRQ Type	0	1	This should be set to 0x80 to indicate a valid CRQ event.
VNIC Command	1	1	This field will be REQUEST_DUMP.
Reserved	2	2	This field is reserved, and should be set to 0.
IOBA	4	4	This field is an I/O bus address referring to a TCE-mapped buffer used by system firmware to place the VNIC dump.
Length	8	4	This field contains the length of the VNIC dump buffer.
Reserved	12	4	This field is reserved and should be set to 0.

Table 324. REQUEST_DUMP_RSP Command

Field Name	Byte Offset	Length	Definition
CRQ Type	0	1	This should be set to 0x80 to indicate a valid CRQ event.
VNIC Command	1	1	This field will be REQUEST_DUMP_RSP.
Reserved	2	6	This field is reserved, and should be set to 0.
Dumped Length	8	4	This field contains the amount of data placed into the dump buffer.
Return Code	12	4	This is a return code for the operation as defined in Table 305, “VNIC Return Code,” on page 846.

K.6.8 Reliability, Availability, and Service (RAS) Support

The VNIC RAS support allows the tracing of information within system firmware, and control of consistency checking done by firmware. Individual components of firmware will be exposed to the VNIC Client, and each component can independently have their tracing and error checking levels increased and decreased. Each individual component’s trace information can be collected independently from others.

Trace entries will be returned to the VNIC client in timebase order.

The upper 16 bits of the trace ID for the Firmware Trace Data Format are an AIX RAS tracehook ID, and the lower 16 bits are an AIX RAS subhookid.

Prior to a successful LOGIN request, all components related to the VNIC may not be available in the list of components. To get a complete list of all possible components, the RAS commands should be delayed until after a successful LOGIN unless a pre-LOGIN problem is being diagnosed.

The CONTROL_RAS command can be used to resize the individual components’ trace buffers, but due to the limited memory available in the system firmware, increasing the sizes of one trace buffer may require decreasing the size of a different component’s trace buffer.

The REQUEST_RAS_COMP_NUM and REQUEST_RAS_COMP_NUM_RSP commands are defined in Table 325, “REQUEST_RAS_COMP_NUM and REQUEST_RAS_COMP_NUM_RSP Commands,” on page 866, and the REQUEST_RAS_COMPS and REQUEST_RAS_COMPS_RSP command format is defined in Table 326,

"REQUEST_RAS_COMPS and REQUEST_RAS_COMPS_RSP Commands," on page 866. The COLLECT_FW_TRACE and COLLECT_FW_TRACE_RSP commands are defined in Table 328, "COLLECT_FW_TRACE and COLLECT_FW_TRACE_RSP Commands," on page 867.

Table 325. REQUEST_RAS_COMP_NUM and REQUEST_RAS_COMP_NUM_RSP Commands

Field Name	Byte Offset	Length	Definition
CRQ Type	0	1	This should be set to 0x80 to indicate a valid CRQ event.
VNIC Command	1	1	This field will be REQUEST_RAS_COMP_NUM or REQUEST_RAS_COMP_NUM_RSP.
Reserved	2	2	This field is reserved, and should be set to 0.
Number of Components	4	4	This field contains the number of individual firmware components whose RAS characteristics can be independently modified.
Reserved	8	4	This field is reserved, and should be set to 0.
Return Code	12	4	On a response, this field will contain a return code for the request as defined in Table 305, "VNIC Return Code," on page 846. This field is reserved for a request.

.

Table 326. REQUEST_RAS_COMPS and REQUEST_RAS_COMPS_RSP Commands

Field Name	Byte Offset	Length	Definition
CRQ Type	0	1	This should be set to 0x80 to indicate a valid CRQ event.
VNIC Command	1	1	This field will be REQUEST_RAS_COMPS or REQUEST_RAS_COMPS_RSP.
Reserved	2	2	This field is reserved, and should be set to 0.
IOBA	4	4	This field contains an I/O bus address of a TCE-mapped buffer containing an array of Firmware Component structures as defined in Table 330, "Firmware Component Format," on page 868. The VNIC client should ensure the buffer is large enough to contain the number of components as returned in a REQUEST_RAS_COMP_NUM_RSP command.
Length	8	4	This field is the length of the buffer referred to by the IOBA field. It should be some multiple of the size of the Firmware Component format.
Return Code	12	4	This is a return code for the operation as defined in Table 305, "VNIC Return Code," on page 846.

Table 327. CONTROL_RAS and CONTROL_RAS_RSP Commands

Field Name	Byte Offset	Length	Definition
CRQ Type	0	1	This should be set to 0x80 to indicate a valid CRQ event.
VNIC Command	1	1	This field will be CONTROL_RAS or CONTROL_RAS_RSP.

Table 327. CONTROL_RAS and CONTROL_RAS_RSP Commands ***(Continued)***

Field Name	Byte Offset	Length	Definition
Correlator	2	1	This field contains a Correlator for a Firmware Component as defined in Table 330, “Firmware Component Format,” on page 868 that this command should act on.
Level	3	1	This value should be a value between 0 and 9, where a larger number indicates a higher detail of tracing or error checking.
Operation	4	1	This field controls what action the CONTROL_RAS command performs. If this value is a 1, use the Level field to modify the current trace level of the specified component. If this value is a 2, use the Level field to modify the current error checking level of the specified component. If this value is a 3, suspend the tracing for the specified component that was previously on. If this value is a 4, resume the tracing for the specified component that was previously suspended. If this value is a 5, turn tracing for the specified component on. If this value is a 6, turn tracing for the specified component off. If this value is a 7, change the size of the specified trace buffer for the specified component. All other values are reserved.
Trace Buffer Size	5	3	If Operation is a 7, this field contains the requested size of the specified trace buffer. On a response, will be filled in with the current size of the trace buffer. For all other Operation values, this field is reserved.
Reserved	8	4	This field is reserved, and should be set to 0.
Return Code	12	4	This is a return code for the operation as defined in Table 305, “VNIC Return Code,” on page 846.

Table 328. COLLECT_FW_TRACE and COLLECT_FW_TRACE_RSP Commands

Field Name	Byte Offset	Length	Definition
CRQ Type	0	1	This should be set to 0x80 to indicate a valid CRQ event.
VNIC Command	1	1	This field will be COLLECT_FW_TRACE or COLLECT_FW_TRACE_RSP.
Correlator	2	1	This field contains a Correlator for a Firmware Component as defined in Table 330, “Firmware Component Format,” on page 868 that this command should act on.
Reserved	3	1	This field is reserved, and should be set to 0.
IOBA	4	4	This field contains the I/O bus address of a TCE-mapped buffer of the indicated size that will be used by firmware to return the trace.
Buffer Length	8	4	This field contains the length of the buffer in bytes used to collect the trace information. On a COLLECT_FW_RSP, this value will indicate how much trace data is actually placed in the buffer. The trace data is an array of entries with the format as defined in Table 329, “Firmware Trace Data Entry Format,” on page 868.
Return Code	12	4	This is a return code for the operation as defined in Table 305, “VNIC Return Code,” on page 846.

Table 329. Firmware Trace Data Entry Format

Field Name	Byte Offset	Length
Trace ID	0	4
Number Valid Trace Data	4	1
Reserved	5	3
PMC Registers	8	8
Timebase	16	8
Trace Data 1	24	8
Trace Data 2	32	8
Trace Data 3	40	8
Trace Data 4	48	8
Trace Data 5	56	8

Table 330. Firmware Component Format

Field Name	Byte Offset	Length	Definition
Component Name	0	48	This field contains an ASCII string containing a readable name of the component.
Trace Buffer Size	48	4	This field contains the size of the trace buffer.
Correlator	52	1	This field contains a value to be used on a COLLECT_FW_TRACE or CONTROL_RAS command to identify which component to operate on.
Trace Level	53	1	This field shows the current trace level, as defined in Table 327, “CONTROL_RAS and CONTROL_RAS_RSP Commands,” on page 866. A value of 0xFF indicates this component does not support tracing.
Parent Correlator	54	1	This field contains the correlator of the parent component. If this value is 0xFF, there is no parent.
Error Checking	55	1	This field contains the error checking level for this component. It contains a value from 0-9, where 0 means no extra error checking, and 9 means the highest level of consistency checking. A value of 0xFF indicates this component does not support changing its level of error checking.
Trace State	56	1	If this field is a 0, the component’s tracing is turned off. If this field is a 1, the component’s tracing is turned on.
Reserved	57	7	This field is reserved, and should be set to 0.
Description	64	192	This field contains an ASCII string containing a readable description of the component.

K.6.9 Statistics Support

The REQUEST_STATISTICS command as defined in Table 331, “REQUEST_STATISTICS Command,” on page 869 is used by the VNIC client to obtain statistic counters kept by system firmware and the physical adapter supporting the VNIC.

The REQUEST_STATISTICS_RSP command is defined in Table 332, “REQUEST_STATISTICS_RSP Command,” on page 869.

In the event a given VNIC does not support the retrieval of certain of the statistics, the statistic will have a -1 value returned in it.

The REQUEST_DEBUG_STATS command defined in Table 334, “REQUEST_DEBUG_STATS command,” on page 871 is used by the VNIC client to retrieve an unarchitected block of statistics that is implementation dependent which may be used to debug firmware problems. This is an optional command, and the actual data returned may vary from implementation to implementation.

Table 331. REQUEST_STATISTICS Command

Field Name	Byte Offset	Length	Definition
CRQ Type	0	1	This should be set to 0x80 to indicate a valid CRQ event.
VNIC Command	1	1	This field will be REQUEST_STATISTICS.
Flags	2	1	Bit 0: If set, retrieve the physical port statistics. If the VNIC doesn’t have authority to retrieve the physical port statistics, the command may fail. If this bit is 0, retrieve the logical port statistics. Bit 1: If this field is set to 1, clear the statistics. If this field is set to 0, do not clear any statistics. Bit 2-7: This fields are reserved, and should be set to 0.
Reserved	3	1	This field is reserved, and should be set to 0.
IOBA	4	4	This field is an I/O bus address referring to a TCE-mapped buffer used by system firmware to place the VNIC statistics block as defined in Table 333, “VNIC Statistics Version 1,” on page 870.
Length	8	4	This field contains the length of the VNIC statistics buffer.
Reserved	12	4	This field is reserved and should be set to 0.

Table 332. REQUEST_STATISTICS_RSP Command

Field Name	Byte Offset	Length	Definition
CRQ Type	0	1	This should be set to 0x80 to indicate a valid CRQ event.
VNIC Command	1	1	This field will be REQUEST_STATISTICS_RSP.
Reserved	2	2	This field is reserved, and should be set to 0.
Reserved	4	8	This field is reserved and should be set to 0.
Return Code	12	4	This is a return code for the operation as defined in Table 305, “VNIC Return Code,” on page 846.

Table 333. VNIC Statistics Version 1

Field Name	Byte Offset	Length
Version	0	4
Promiscuous	4	4
Received Packets	8	8
Bytes Received	16	8
Packets Sent	24	8
Bytes Sent	32	8
Unicast Packets Sent	40	8
Unicast Packets Received	48	8
Multicast Packets Sent	56	8
Multicast Packets Received	64	8
Broadcast Packets Sent	72	8
Broadcast Packets Received	80	8
Alignment Errors	88	8
FCS Errors	96	8
Single Collision Frames	104	8
Multiple Collision Frames	112	8
SQE Test Errors	120	8
Deferred Transmissions	128	8
Late Collisions	136	8
Excess Collisions	144	8
Internal MAC Transmit Errors	152	8
Carrier Sense	160	8
Too Long Frames	168	8
Internal MAC Receive Errors	176	8
DMA Receive Overrun	184	8
DMA Transmit Underrun	192	8
Receive No Resource	200	8
Too Short Frames	208	8
Reserved	216	40

Table 334. REQUEST_DEBUG_STATS command

Field Name	Byte Offset	Length	Definition
CRQ Type	0	1	This should be set to 0x80 to indicate a valid CRQ event.
VNIC Command	1	1	This field will be REQUEST_DEBUG_STATS or REQUEST_DEBUG_STATS_RSP.
Reserved	2	2	This field is reserved, and should be set to 0.
IOBA	4	4	This field is an I/O bus address referring to a TCE-mapped buffer used by system firmware to place the VNIC debug statistics block.
Length	8	4	This field contains the length of the VNIC statistics buffer. On a REQUEST_DEBUG_STATS_RSP, contains the amount of data filled in.
Return Code	12	4	This is a return code for the operation as defined in Table 305, “VNIC Return Code,” on page 846.

K.6.10 Error Reporting Support

If system firmware encounters an error processing requests related to the physical adapter being virtualized by the VNIC interface, it will generate ERROR_INDICATION commands to the VNIC client, as defined in Table 335, “ERROR_INDICATION Command,” on page 871. The VNIC client may then, at its discretion, obtain detailed error information using the REQUEST_ERROR_INFO command as defined in Table 336, “REQUEST_ERROR_INFO Command,” on page 872. It is the intent that the VNIC client should log the detailed error information using its normal error logging infrastructure and methods.

The REQUEST_ERROR_INFO_RSP command as defined in Table 336, “REQUEST_ERROR_INFO Command,” on page 872 is used by firmware to indicate the successful retrieval of error information. The retrieval of detailed error information allows firmware to reuse the resources for tracking that error. Detailed error information can only be requested for a specific error once.

If system firmware encounters an error while the VNIC client is not connected, firmware will log the detailed error information using firmware error logging methods.

Firmware will have a finite amount of space reserved for storing detailed error information. In some situations, some detailed error information may be unavailable in response to a REQUEST_ERROR_INFO command if too many errors are being logged in firmware. If the detailed error information is overwritten prior to the VNIC client performing the relative REQUEST_ERROR_INFO command, an error return code will be returned.

If the fatal error bit is set, the VNIC firmware has encountered a fatal error preventing it from automatically recovering from the error. The VNIC client should use the RAS facilities to collect any error information, collect any RAS tracing, statistics, and possibly a dump. Once all available error data has been collected, it is the VNIC client’s responsibility to cause the VNIC to restart. This can be accomplished in one of two ways: by freeing its CRQ using H_FREE_CRQ, or by initiating a VNIC reset using the H_VIOCTL FW_RESET subfunction.

Table 335. ERROR_INDICATION Command

Field Name	Byte Offset	Length	Definition
CRQ Type	0	1	This should be set to 0x80 to indicate a valid CRQ event.

Table 335. ERROR_INDICATION Command ***(Continued)***

Field Name	Byte Offset	Length	Definition
VNIC Command	1	1	This field will be ERROR_INDICATION.
Flags	2	1	Bit 0: If this bit is 1, this is a fatal error. Bit 1-7: These fields are reserved, and should be set to 0.
Reserved	3	1	This field is reserved, and should be set to 0.
Error Identification	4	4	This field is set to the error identification number that can be used to retrieve detailed information about the error using the REQUEST_ERROR_INFO command. If this value is set to 0, there is no more detailed error information to retrieve, and the ERROR_INDICATION contains all relevant information.
Detailed Error Size	8	4	This field contains the size of the detailed error information associated with the error.
Error Cause	12	2	This field contains a value as detailed in Table 338, “Error Cause,” on page 873 showing the cause of the error.
Reserved	14	2	This field is reserved, and should be set to 0.

Table 336. REQUEST_ERROR_INFO Command

Field Name	Byte Offset	Length	Definition
CRQ Type	0	1	This should be set to 0x80 to indicate a valid CRQ event.
VNIC Command	1	1	This field will be REQUEST_ERROR_INFO.
Reserved	2	2	This field is reserved, and should be set to 0.
Buffer IOBA	4	4	This field contains the I/O bus address of a TCE-mapped buffer to be used by system firmware to write the detailed error information into.
Buffer Length	8	4	This field contains the length of the TCE-mapped buffer.
Error Identification	12	4	This field contains the error identification from an ERROR_INDICATION command that specifies which detailed error information to obtain.

Table 337. REQUEST_ERROR_INFO_RSP Command

Field Name	Byte Offset	Length	Definition
CRQ Type	0	1	This should be set to 0x80 to indicate a valid CRQ event.
VNIC Command	1	1	This field will be REQUEST_ERROR_INFO_RSP.
Reserved	2	2	This field is reserved, and should be set to 0.

Table 337. REQUEST_ERROR_INFO_RSP Command ***(Continued)***

Field Name	Byte Offset	Length	Definition
Error Identification	4	4	This field contains the error identification from an ERROR_INDICATION command. This field can be used to correlate this response to s REQUEST_ERROR_INFO command, allowing multiple requests for errors to be outstanding at the same time.
Length	8	4	This field contains the length of data successfully returned in the TCE-mapped buffer.
Return Code	12	4	This is a return code for the operation as defined in Table 305, “VNIC Return Code,” on page 846.

Table 338. Error Cause

Value	Definition
Adapter Problem	0
Bus Problem	1
Firmware Problem	2
Device Driver Problem	3
EEH Recovery	4
Firmware Updated	5
Low Memory	6
Reserved	7-65535

K.6.11 Link State Change

This LINK_STATE_INDICATION command as defined in Table 339, “LINK_STATE_INDICATION Command,” on page 873 is an unacknowledged command sent by system firmware to inform the VNIC client when the state of the link changes. The VNIC client can also use QUERY_PHYS_PARMS at any time to poll for link state changes.

Table 339. LINK_STATE_INDICATION Command

Field Name	Byte Offset	Length	Definition
CRQ Type	0	1	This should be set to 0x80 to indicate a valid CRQ event.
VNIC Command	1	1	This field will be LINK_STATE_INDICATION.
Reserved	2	2	This field is reserved, and should be set to 0.
Physical Link State	4	1	If this field is a 0, the physical link is down, if the field is a 1, the physical link is up.

Table 339. LINK_STATE_INDICATION Command ***(Continued)***

Field Name	Byte Offset	Length	Definition
Logical Link State	5	1	If this field is a 0, the logical link is down and the VNIC cannot communicate with other VNICs on the same adapter, if the field is a 1, the logical link is up, and the VNIC can communicate with other VNICs on the same adapter.
Reserved	6	10	This field is reserved, and should be set to 0.

K.6.12 Change MAC Address

The CHANGE_MAC_ADDR command defined in Table 340, “CHANGE_MAC_ADDR and CHANGE_MAC_ADDR_RSP Commands,” on page 874 allows the VNIC client to change the current MAC address. The request to change may fail due to Access Control List entries set up by the administrator.

Table 340. CHANGE_MAC_ADDR and CHANGE_MAC_ADDR_RSP Commands

Field Name	Byte Offset	Length	Definition
CRQ Type	0	1	This should be set to 0x80 to indicate a valid CRQ event.
VNIC Command	1	1	This field will be CHANGE_MAC_ADDR or CHANGE_MAC_ADDR_RSP.
MAC Address	2	6	This field contains the new requested MAC address on a CHANGE_MAC_ADDR command, and the current MAC address on a CHANGE_MAC_ADDR_RSP.
Reserved	8	4	This field is reserved, and should be set to 0.
Return Code	12	4	This is a return code for the operation as defined in Table 305, “VNIC Return Code,” on page 846.

K.6.13 Multicast Support

The MULTICAST_CTRL command defined in Table 341, “MULTICAST_CTRL and MULTICAST_CTRL_RSP Commands,” on page 875 allows the VNIC client to manage the reception of Multicast Ethernet traffic. Individual multicast MAC addresses may be enabled and disabled, as well as all multicast traffic.

The VNIC client can choose to enable more than the maximum unique multicast Ethernet addresses as returned in the Capabilities exchange. In the event the VNIC client does so, system firmware will either enable the MAC address via a non-exact hashing multicast reception mechanism if the hardware supports it, or will enable all multicast addresses. When this is done, system firmware will report exact matches through the unique multicast Ethernet filter via the Exact Match bit defined in the Receive Completion Descriptor as defined in Table 356, “Receive Completion Descriptor,” on page 883. If the Exact Match bit is off, and a multicast packet was returned in the Receive Completion Descriptor, the multicast packet either matches a non-exact hashing mechanism if one exists or system firmware has enabled all multicast MAC address reception.

Table 341. MULTICAST_CTRL and MULTICAST_CTRL_RSP Commands

Field Name	Byte Offset	Length	Definition
CRQ Type	0	1	This should be set to 0x80 to indicate a valid CRQ event.
VNIC Command	1	1	This field will be MULTICAST_CTRL or MULTICAST_CTRL_RSP.
MAC Address	2	6	This field contains the new requested multicast MAC address, as appropriate for the specific options requested.
Flags	8	1	Bit 0: Enable specified multicast MAC address Bit 1: Disable specified multicast MAC address Bit 2: Enable the reception of all multicast MAC addresses. This does not affect the multicast addresses enabled through Bit 0. Bit 3: Disable the reception of all multicast MAC addresses. This does not affect the multicast addresses enabled through Bit 0. Bit 4-7: These bits are reserved, and should be set to 0.
Reserved	9	3	This field is reserved, and should be set to 0.
Return Code	12	4	This is a return code for the operation as defined in Table 305, “VNIC Return Code,” on page 846.

K.6.14 VPD Support

The VPD commands may be used by the VNIC client to collect, store, and display VPD related to the physical adapter backing the VNIC. As the exact adapter may change during partition mobility operations, it is suggested this data not be relied upon operationally, and be used with the understanding that it may change from request to request.

The VPD commands are defined in Table 342, “GET_VPD_SIZE Command,” on page 875, Table 343, “GET_VPD_SIZE_RSP Command,” on page 875, Table 344, “GET_VPD Command,” on page 876, and Table 345, “GET_VPD_RSP Command,” on page 876.

Table 342. GET_VPD_SIZE Command

Field Name	Byte Offset	Length	Definition
CRQ Type	0	1	This should be set to 0x80 to indicate a valid CRQ event.
VNIC Command	1	1	This field will be GET_VPD_SIZE.
Reserved	2	14	This field is reserved, and should be set to 0.

Table 343. GET_VPD_SIZE_RSP Command

Field Name	Byte Offset	Length	Definition
CRQ Type	0	1	This should be set to 0x80 to indicate a valid CRQ event.
VNIC Command	1	1	This field will be GET_VPD_SIZE_RSP.
Reserved	2	2	This field is reserved, and should be set to 0.

Table 343. GET_VPD_SIZE_RSP Command ***(Continued)***

Field Name	Byte Offset	Length	Definition
Length	4	8	This field contains the length of the VPD present.
Return Code	12	4	This is a return code for the operation as defined in Table 305, “VNIC Return Code,” on page 846.

Table 344. GET_VPD Command

Field Name	Byte Offset	Length	Definition
CRQ Type	0	1	This should be set to 0x80 to indicate a valid CRQ event.
VNIC Command	1	1	This field will be GET_VPD.
Reserved	2	2	This field is reserved, and should be set to 0.
IOBA	4	4	This field is an I/O bus address referring to a TCE-mapped buffer used by system firmware to place the VNIC VPD.
Length	8	4	This field contains the length of the VPD buffer.
Reserved	12	4	This field is reserved and should be set to 0.

Table 345. GET_VPD_RSP Command

Field Name	Byte Offset	Length	Definition
CRQ Type	0	1	This should be set to 0x80 to indicate a valid CRQ event.
VNIC Command	1	1	This field will be GET_VPD_RSP.
Reserved	2	10	This field is reserved, and should be set to 0.
Return Code	12	4	This is a return code for the operation as defined in Table 305, “VNIC Return Code,” on page 846.

K.6.15 Access Control Support

The VNIC may have certain Access Control Lists (ACLs) in effect, and some of these may change dynamically. The ACL_CHANGE_INDICATION command defined in Table 346, “ACL_CHANGE_INDICATION,” on page 877 is sent by system firmware to the VNIC client in the event any of the ACLs have changed dynamically.

The ACL_QUERY command defined in Table 347, “ACL_QUERY,” on page 877 and its associated response defined in Table 348, “ACL_QUERY_RSP,” on page 877 may be used by the VNIC client to obtain information about the ACLs in effect to enable earlier error checking or ease of use functions.

Table 346. ACL_CHANGE_INDICATION

Field Name	Byte Offset	Length	Definition
CRQ Type	0	1	This should be set to 0x80 to indicate a valid CRQ event.
VNIC Command	1	1	This field will be ACL_CHANGE_INDICATION.
Change Type	2	2	If this field is a 0, the MAC address ACLs have changed. If this field is a 1, the VLAN id ACLs have changed. All other values are reserved.
Reserved	4	12	This field is reserved, and should be set to 0.

Table 347. ACL_QUERY

Field Name	Byte Offset	Length	Definition
CRQ Type	0	1	This should be set to 0x80 to indicate a valid CRQ event.
VNIC Command	1	1	This field will be ACL_QUERY.
Reserved	2	2	This field is reserved, and should be set to 0.
IOBA	4	4	This field is an I/O bus address referring to a TCE-mapped buffer used by system firmware to place the ACL information. Upon reception of a ACL_QUERY_RSP with a Success return code, this buffer will be filled in with the structure as defined in Table 349 on page 877
Size	8	4	This field contains the size of the buffer mapped by the IOBA.
Reserved	12	4	This field is reserved, and should be set to 0.

Table 348. ACL_QUERY_RSP

Field Name	Byte Offset	Length	Definition
CRQ Type	0	1	This should be set to 0x80 to indicate a valid CRQ event.
VNIC Command	1	1	This field will be ACL_QUERY_RSP.
Reserved	2	10	This field is reserved, and should be set to 0.
Return Code	12	4	This is a return code for the operation as defined in Table 305, “VNIC Return Code,” on page 846.

Table 349. ACL Buffer

Field Name	Byte Offset	Length	Definition
Total Length	0	4	This field is the total length of the ACL Buffer.

Table 349. ACL Buffer ***(Continued)***

Field Name	Byte Offset	Length	Definition
Version	4	4	This field contains the version of ACL Buffer layout. The initial version should be set to 1.
MAC ACLs in effect	8	1	This field contains a 1 if there are MAC restrictions in effect, zero otherwise.
VLAN id ACLs in effect	9	1	This field contains a 1 if there are VLAN id ACLs in effect, zero otherwise.
Reserved	10	22	This field is reserved, and should be set to 0.
Number of Allowed MAC addresses	32	4	This field contains the number of allowable MAC addresses.
Offset to array of allowed MAC addresses	36	4	This field contains an offset from the start of the ACL Buffer to an array of six byte MAC addresses.
Number of allowed VLAN ids	40	4	This field contains the number of allowable VLAN ids.
Offset to array of allowed VLAN ids	44	4	This field contains an offset from the start of the ACL buffer to an array of 2 byte VLAN ids.
Reserved	48	80	This field is reserved, and should be set to 0.
Array of allowed MAC addresses	variable	variable	This is the array of six byte MAC addresses, with a size as defined in the Number of Allowed MAC address field.
Array of allowed VLAN ids	variable	variable	This is the array of two byte VLAN ids, with a size as defined in the Number of Allowed VLAN ids field.

K.6.16 Debugging Support

The TUNE command defined in Table 350, “TUNE Command,” on page 878 may be used by the VNIC client to opaquely pass tuning data from the VNIC client to system firmware. As the exact firmware backing a VNIC client may change during partition mobility operations, it is suggested this data not be relied upon operationally, and be used with the understanding that it may change from adapter to adapter.

A TUNE_RSP command defined in Table 351, “TUNE_RSP Command,” on page 879 will be generated by system firmware upon completion of the TUNE command.

This command is an optional VNIC command, and may not be supported for all VNIC implementations or versions of system firmware.

Table 350. TUNE Command

Field Name	Byte Offset	Length	Definition
CRQ Type	0	1	This should be set to 0x80 to indicate a valid CRQ event.
VNIC Command	1	1	This field will be TUNE.
Reserved	2	2	This field is reserved, and should be set to 0.
IOBA	4	4	This field is an I/O bus address referring to a TCE-mapped buffer used by system firmware to obtain the tuning parameters.
Length	8	4	This field contains the length of the VPD buffer.

Table 350. TUNE Command ***(Continued)***

Field Name	Byte Offset	Length	Definition
Reserved	12	4	This field is reserved and should be set to 0.

Table 351. TUNE_RSP Command

Field Name	Byte Offset	Length	Definition
CRQ Type	0	1	This should be set to 0x80 to indicate a valid CRQ event.
VNIC Command	1	1	This field will be TUNE_RSP.
Reserved	2	10	This field is reserved, and should be set to 0.
Return Code	12	4	This is a return code for the operation as defined in Table 305, “VNIC Return Code,” on page 846.

K.7 Subordinate CRQ Definitions

Frame transmission and reception is handled through the Subordinate CRQ infrastructure, using the H_SEND_SUB_CRQ and H_SEND_SUB_CRQ_INDIRECT hypervisor calls.

K.7.1 Frame Transmission

Since each Transmit Completion Sub-CRQ is tied to a specific Transmit Submission Sub-CRQ, the Transmit Descriptor correlator must only be unique for a given Transmit Completion Sub-CRQ.

Several versions of Transmit Descriptors exist. Each version has a Descriptor Version byte at byte offset one in the descriptor, which specifies the layout of the later thirty bytes. A sorted array is returned in the LOGIN response specifying all versions of transmit descriptor supported by the VNIC. The versions of the transmit descriptor offering the best performance appear in the array first. All VNIC versions will support Transmit Descriptor Version Zero defined in Table 352, “Transmit Descriptor Version Zero,” on page 879, but that version may not offer the best performance.

Transmit Descriptor Version Two defined in Table 355, “Transmit Descriptor Version Two,” on page 882 is designed to be used in combination with a previous use of Transmit Descriptor Version Zero or Transmit Descriptor Version One defined in Table 354, “Transmit Descriptor Version One,” on page 881

Table 352. Transmit Descriptor Version Zero

Field Name	Byte Offset	Length	Definition
Sub-CRQ Format	0	1	This value should be set to 0x80 to indicate a valid Sub-CRQ event.
Descriptor Version	1	1	This field is 0 for a Version Zero Transmit Descriptor

Table 352. Transmit Descriptor Version Zero ***(Continued)***

Field Name	Byte Offset	Length	Definition
Flags	2	1	Bit 0: If set to 1, this frame should use the large send offload feature of the physical adapter, assuming it was previously enabled through the use of CONTROL_IP_OFFLOAD command. Bit 1: If set to 1, this frame should use the IP checksum feature of the physical adapter, assuming it was previously enabled through the use of CONTROL_IP_OFFLOAD command. Bit 2: If set to 1, this frame should use the TCP checksum feature of the physical adapter, assuming it was previously enabled through the use of CONTROL_IP_OFFLOAD command. Bit 3: If set to 1, this frame should use the physical adapter's capability of inserting VLAN headers, using the VLAN header field as the source for the values to insert. Bit 4:If set to 1, this frame should use the UDP checksum feature of the physical adapter, assuming it was previously enabled through the use of CONTROL_IP_OFFLOAD command. Bit 5: If set to 1, this frame spans multiple transmit descriptors. Bit 6: If set to 1, this frame contains the last fragment of a complete packet for transmission. Bit 7:If set to 1, this frame requires a transmit completion event to be posted to the VNIC client's Transmit Completion Sub-CRQ. If set to 0, no completion event will be generated unless an error occurred.
IP Header offset	3	1	Bit 0:If set to zero, this frame contains an IPv4 frame. If set to 1, this frames contains an IPv6 frame. Bits 1-7: This field should be set to the offset of IP header in the first descriptor of a chain if any checksum offload or large send offload is enabled. If neither checksum offload nor large send offload are enabled, this should be set to 0.
TCP/UDP Header offset or IP Data offset	4	2	In the first descriptor of a chain, if any checksum offload or large send offload function should be done for this packet, this field must be set to the offset of the first byte of data after the IP header and extension headers. If no checksum offload or large send offload function should be done for this packet, this field must be set to 0.
VLAN Header	6	2	If VLAN header insertion has been enabled, this field contains the VLAN header to be inserted if indicated in the Flags byte.
Reserved	8	1	This field is reserved and should be set to 0.
MSS Size	9	3	If large send offload is enabled, this field contains the MSS size.
Correlator	12	4	This field is set to a unique opaque value generated by the VNIC client that allows the device driver to correlate a transmit submission to an eventual completion. This value must be set even if the Completion Required bit is set to 0 in the event an error happens.
IOBA1	16	4	This field contains an I/O bus address valid for the VNIC device that refers to the first chunk of the transmit frame for this descriptor.
Length1	20	4	This contains the length of the frame fragment pointed to by IOBA1.
IOBA2	24	4	This field contains an I/O bus address valid for the VNIC device that refers to the second fragment of the transmit frame for this descriptor. If the corresponding length field is 0, this field is ignored.
Length2	28	4	This contains the length of the frame fragment pointed to by IOBA2. If IOBA2 is invalid, this field should be set to 0.

Table 353. Transmit Completion Descriptor

Field Name	Byte offset	Length	Definition
Sub-CRQ Format	0	1	This value should be set to 0x80 to indicate a valid Sub-CRQ event.
Number of Completions	1	1	This is the number of return code and correlator pairs of this descriptor that are valid. It must be a value from 1 to 5.
Return Codes	2	10	This is an array of 5 two byte integer return codes as defined in Table 307, “VNIC Architected Return Values,” on page 849.
Correlators	12	20	This is an array of five four-byte correlator values as taken from the Transmit Submission Descriptor.

Table 354. Transmit Descriptor Version One

Field Name	Byte Offset	Length	Definition
Sub-CRQ Format	0	1	This value should be set to 0x80 to indicate a valid Sub-CRQ event.
Descriptor Version	1	1	This field is 1 for a Version One Transmit Descriptor
Flags	2	1	Bit 0: If set to 1, this frame should use the large send offload feature of the physical adapter, assuming it was previously enabled through the use of CONTROL_IP_OFFLOAD command. Bit 1: If set to 1, this frame should use the IP checksum feature of the physical adapter, assuming it was previously enabled through the use of CONTROL_IP_OFFLOAD command. Bit 2: If set to 1, this frame should use the TCP checksum feature of the physical adapter, assuming it was previously enabled through the use of CONTROL_IP_OFFLOAD command. Bit 3: If set to 1, this frame should use the physical adapter’s capability of inserting VLAN headers, using the VLAN header field as the source for the values to insert. Bit 4:If set to 1, this frame should use the UDP checksum feature of the physical adapter, assuming it was previously enabled through the use of CONTROL_IP_OFFLOAD command. Bit 5: If set to 1, this frame spans multiple transmit descriptors. Bit 6: If set to 1, this frame contains the last fragment of a complete packet for transmission. Bit 7:If set to 1, this frame requires a transmit completion event to be posted to the VNIC client’s Transmit Completion Sub-CRQ. If set to 0, no completion event will be generated unless an error occurred.
IP Header offset	3	1	Bit 0:If set to zero, this frame contains an IPv4 frame. If set to 1, this frames contains an IPv6 frame. Bits 1-7: This field should be set to the offset of IP header in the first descriptor of a chain if any checksum offload or large send offload is enabled. If neither checksum offload nor large send offload are enabled, this should be set to 0.
TCP/UDP Header offset or IP Data offset	4	2	In the first descriptor of a chain, if any checksum offload or large send offload function should be done for this packet, this field must be set to the offset of the first byte of data after the IP header and extension headers. If no checksum offload or large send offload function should be done for this packet, this field must be set to 0.

Table 354. Transmit Descriptor Version One ***(Continued)***

Field Name	Byte Offset	Length	Definition
VLAN Header	6	2	If this frame is a VLAN-tagged frame, this field contains the VLAN tag even if it is already present in the frame, and even if VLAN offload is disabled. If already present in the frame, this merely provides a hint to enable fast transmission of this frame. If the VLAN header is not present in the frame, this field contains the VLAN header to be inserted if indicated in the Flags byte.
Reserved	8	1	This field is reserved and should be set to 0.
MSS Size	9	3	If large send offload is enabled, this field contains the MSS size.
Correlator	12	4	This field is set to a unique opaque value generated by the VNIC client that allows the device driver to correlate a transmit submission to an eventual completion. This value must be set even if the Completion Required bit is set to 0 in the event an error happens.
IOBA1	16	4	This field contains an I/O bus address valid for the VNIC device that refers to the first chunk of the transmit frame for this descriptor.
Length1	20	4	This contains the length of the frame fragment pointed to by IOBA1.
Destination MAC address	24	6	This field contains the destination MAC address as specified in the frame to be sent. The frame still must contain this information; this merely provides a hint to enable fast transmission of this frame.
Ethertype	30	2	This field contains a copy of the ethertype from the specified frame to be sent. The frame still must contain this information; this merely provides a hint to enable fast transmission of this frame.

Table 355. Transmit Descriptor Version Two

Field Name	Byte Offset	Length	Definition
Sub-CRQ Format	0	1	This value should be set to 0x80 to indicate a valid Sub-CRQ event.
Descriptor Version	1	1	This field is 2 for a Version Two Transmit Descriptor. This transmit descriptor should be used following another valid Transmit Descriptor Format such as Version Zero or Version One. It inherits any advanced features from the previous Transmit Descriptor.
Flags	2	1	Bit 0-5: These bits are reserved, and should be set to 0. Bit 6: If set to 1, this frame contains the last fragment of a complete packet for transmission. Bit 7:If set to 1, this frame requires a transmit completion event to be posted to the VNIC client's Transmit Completion Sub-CRQ. If set to 0, no completion event will be generated unless an error occurred.
Reserved	3	1	This field is reserved, and should be set to 0.
IOBA1	4	4	This field contains an I/O bus address valid for the VNIC device that refers to the first chunk of the transmit frame for this descriptor.
Length1	8	4	This field contains the length of the frame fragment pointed to by IOBA1.
Correlator	12	4	This field is set to a unique opaque value generated by the VNIC client that allows the device driver to correlate a transmit submission to an eventual completion. This value must be set even if the Completion Required bit is set to 0 in the event an error happens.

Table 355. Transmit Descriptor Version Two ***(Continued)***

Field Name	Byte Offset	Length	Definition
IOBA2	16	4	This field contains an I/O bus address valid for the VNIC device that refers to the second chunk of the transmit frame for this descriptor.
Length2	20	4	This contains the length of the frame fragment pointed to by IOBA2.
IOBA3	24	4	This field contains an I/O bus address valid for the VNIC device that refers to the third chunk of the transmit frame for this descriptor.
Length3	28	4	This contains the length of the frame fragment pointed to by IOBA3.

K.7.2 Frame Reception

Multiple Receive Buffer Add Sub-CRQs can be configured to allow the VNIC client to efficiently allocate receive buffers of different sizes. In the event multiple Sub-CRQs are allocated for this purpose, it is the VNIC client's responsibility to always allocate the receive buffer size for the Receive Buffer Add Sub-CRQs that are returned by system firmware as defined in Table 314, "LOGIN Response Buffer," on page 858.

System firmware will configure the correct buffer sizes based on the current VNIC maximum transmission unit, current number of Receive Buffer Add Sub-CRQs, and physical adapter capabilities. In all cases, all receive buffers given to an individual Receive Buffer Add Sub-CRQ must be of the same size.

Since a Receive Buffer Correlator may appear on only a single Receive Completion Sub-CRQ, the Receive Buffer Correlators must be unique for a given Receive Completion Sub-CRQ.

Since every buffer added to all Receive Buffer Add Sub-CRQs associated with a given Receive Completion Sub-CRQ could be received simultaneously, each Receive Completion Sub-CRQ should be sized to handle every possible buffer given to system firmware on its associated Receive Buffer Add Sub-CRQs.

Some implementations of VNIC devices may have alignment requirements. To ensure efficient use of receive buffers, VNIC clients are encouraged to use at least cache-line aligned receive buffers.

.

Table 356. Receive Completion Descriptor

Field Name	Byte Offset	Length	Definition
Sub-CRQ Format	0	1	This should be set to 0x80 to indicate a valid Sub-CRQ event.
Flags	1	1	Bit 0: If this bit is a 1, it indicates system firmware has validated the IP checksum field in the referenced packet was verified to be good. Bit 1: If this bit is a 1, it indicates system firmware has validated the TCP/UDP checksum field in the referenced packet was verified to be good. Bit 2: If this bit is a 1, it indicates this frame contains the end of a packet. Bit 3: If this bit is a 1, this packet is an exact match for one of the requested multicast MAC addresses for this VNIC. Bit 4: If this bit is a 1, the TCP/UDP checksum field contains either the complete TCP/UDP checksum or a partial TCP/UDP checksum in the case of an IP fragment packet. Bit 5-7: These bits are reserved, and will be set to 0.
Offset to start of frame data	2	2	This field contains an offset to the start of actual frame data in the returned frame.
Length	4	4	This field contains the length of valid data in this descriptor.

Table 356. Receive Completion Descriptor ***(Continued)***

Field Name	Byte Offset	Length	Definition
Correlator	8	8	This field is the correlator taken from the Receive Buffer Add descriptor that allows the VNIC client to associate this completion with a previously added receive buffer.
TCP/UDP Checksum	16	2	If the TCP/UDP checksum bit is a 1, this field contains either the complete packet's TCP/UDP checksum or a partial TCP/UDP checksum in the event the packet is an IP fragment.
Reserved	18	14	These fields are reserved, and will be set to 0.

Table 357. Receive Buffer Add Descriptor

Field Name	Byte Offset	Length	Definition
Sub-CRQ Format	0	1	This should be set to 0x80 to indicate a valid Sub-CRQ event.
Reserved	1	7	This bytes are reserved, and should be set to 0.
Correlator	8	8	This field is an opaque value that is returned to the VNIC client when the buffer described by this descriptor is used to receive a frame from the network.
IOBA	16	4	This field contains the I/O bus address for the TCE-mapped memory buffer to be used for frame reception.
Length	20	4	This field contains the length of the memory buffer described by this descriptor.
Reserved	24	8	These bytes are reserved, and should be set to 0.

L When to use: Fault vs. Error Log Indicators (Lightpath Mode)

This appendix gives *highly* recommended Service Indicator activation models for typical system issues, when the Lightpath mode is implemented. The purpose of this appendix is to get consistency across platforms, and to answer common questions about how to handle specific issues. The reason that these are recommended rather than required, is due to the range of systems that are involved, specifically related to the different types of physical layouts (for example: deskside, blade and blade chassis, rack-mounted and particularly high end racks).

This appendix does *not* change the architectural requirements specified in other parts of this document, nor the requirement for implementations to support those requirements. If there are any inconsistencies between this appendix and the requirements in the rest of this document, the requirements take precedence over this appendix. It is very important, therefore, that designers understand the requirements in this architecture, and more specifically, those in Chapter 16, "Service Indicators," on page 511.

Table 358, "Service Indicator Activation Models for Typical System Issues (Lightpath Mode)," on page 886 gives the recommended models. The general model, though, is still dictated by the following requirement, copied here from Chapter 2:

R1–16.2.1.1–1. The detector of a fault condition must do the following:

- If the a fault occurs which cannot be isolated appropriately without the user performing some procedure, then activate the Error Log indicator.
- If a fault occurs which can be isolated to a single FRU and if there exists a Fault indicator for the FRU, then activate that FRU Fault indicator, otherwise activate the Error Log indicator.
- If a fault occurs which cannot be isolated to a single FRU and if there exists a Fault indicator for the most likely FRU in the FRU list, then activate that FRU Fault indicator, otherwise activate the Error Log indicator.
- If a fault occurs which is isolated to a group of FRUs (called a FRU group) and if there exists a Fault indicator for each of the FRUs, then activate all the FRU Fault indicators, otherwise activate the Error Log indicator.

Table 358. Service Indicator Activation Models for Typical System Issues (Lightpath Mode)

Component or problem area	Problem or issue	Indicator activation? **(see notes 1, 2)**		Entry made in Service Focal Point error log?	IBM Call Home?	Comments
		FRU Fault indicator? **(see notes 3, 4)**	Error Log indicator? **(see note 4)**			
All	Any not already covered in this table	Consult with the xipSIA architecture team for the proper behavior				
Power supply or fan	Redundant optional one missing	no	no	no	no	This row for information only (this is not a Serviceable Event).
	Redundant non-optional one missing	no	yes	yes	no	
	Failed redundant	yes	no	yes	yes	
	Failed non-redundant	yes	no	yes	yes	
Power regulator not in power supply	Failed	yes	no	yes	yes	Treated the same as any other FRU or component on a FRU which fails.
Power	Insufficient power with all power supplies installed and operational for power domain	no	yes	yes	no	For components that will not power up due to lack of power (for example, a blade in a chassis), and when that component implements a fast blink capability for the green power LED, that component's green power LED remains in the fast blink mode.
	Insufficient power with missing power supply	no	yes	yes	no	
Disabled button pressed	Unable to power on or off component due to power button being disabled	no	no	no	no	These rows for information only (these are not a Serviceable Events).
	KVM or media tray button pressed, but not active because disabled	no	no	no	no	
Temperature detected out of tolerance	Warning only (no performance throttling or shut-down)	no	no	brand dependent	no	
	Performance throttling or shut-down due to over-temp condition	yes	no	yes	no	FRU Fault indicator must be at component that is throttled or shut-down (temperature indicators, which are not architected, do not roll-up to Enclosure Fault indicators).
Memory (DIMMs)	No memory installed at all, or memory that is installed is mismatched	yes	no	yes	no	
	Invalid memory configuration above the base memory (missing memory, mismatched memory, unsupported memory)	no	yes	yes	no	
	Failed or predicted to fail	yes	no	yes	yes	
System VPD	Invalid or missing	yes	no	yes	yes	Lack of some of the required fields in the call-home may result in the call-home being flagged as a lack of entitlement.

Table 358. Service Indicator Activation Models for Typical System Issues (Lightpath Mode) ***(Continued)***

Component or problem area	Problem or issue	Indicator activation? **(see notes 1, 2)**		Entry made in Service Focal Point error log?	IBM Call Home?	Comments
		FRU Fault indicator? **(see notes 3, 4)**	Error Log indicator? **(see note 4)**			
Battery	Missing or failed	yes	no	yes	brand dependent	
	Rechargeable battery needing reconditioning	no	yes	yes	no	
CPU	Failed or predicted to fail	yes	no	yes	yes	
Planar or CEC	Failed	yes	no	yes	yes	
Disk	Failed or predicted to fail	yes	no	yes	yes	
Boot device	Missing boot device	no	yes	yes	no	
	Corrupt image	no	yes	yes	no	
I/O adapter	Fail or non-recoverable EEH error	yes	no	yes	yes	
Service Processor (FSP, BMC, IMM)	Hardware failure	no	yes	yes	yes	
	Firmware failure	no	yes	yes	brand dependent	
BIOS or Flash	Corrupted single side	no	yes	yes	no	
	Both sides corrupted	yes	no	yes	yes	
All	Unisolated event	no	yes	yes	no	See also Requirement R1–16.2.1.1–1
Switch	Failed	yes	no	yes	yes	
System chassis controller (AMM, CME, ITME)	Failed	yes	no	yes	yes	
Midplane	Failed	yes	no	yes	yes	
Cables	Failed or missing	N/A	yes	yes	yes	

Notes:

1. Never activate both a Fault indicator and an Error Log indicator for the same problem. See also Requirement R1–16.2.1.1–1, referenced immediately above Table 358.

2. Fault indicators above the FRU Fault indicator are not specified here, but the requirements specify that a FRU Fault is rolled-up to the next higher level indicator (specifically, the Enclosure Fault indicator).

3. Enclosure Fault indicators and above are only roll-up indicators and are never activated without a FRU Fault indicator being activated. Therefore the column in Table 358 indicates a FRU Fault indicator. That is, if no FRU Fault indicator exists for the problem, then the Error Log indicator is used instead (per Requirement R1–16.2.1.1–1, referenced immediately above Table 358).

4. The activation of the Error Log indicator (previously known as the System Information (Attention) indicator) and Fault indicators are regulated by the following requirements, among others:

R1–16.2.1.1–6. The Error Log indicator must be activated only for Serviceable Events. Serviceable Events are platform, global, regional and local error events that require a service action and possibly a call home when the serviceable event must be handled by a service representative or at least reported to the service provider. Activation of the Error Log indicator notifies the customer of the event and the event indicates to the customer that there must be some intervention to rectify the problem. The intervention may be a service action that the customer can perform or it may require a service provider.

R1–16.2.1.1–5. For each activation of the Error Log and Enclosure Fault Indicators, one of the following must be true:

- If the platform is functional enough to allow it, then an associated entry must be made in an error log that can be queried by a user interface.
- In the case where the platform is not functional enough to allow logging of an error log entry, then there must exist a way for the user to determine the failure associated with the indicator activation (for example, an error code on an op panel on the system).

Bibliography

This section lists documents which were referenced in this specification or which provide additional information, and some useful information for obtaining these documents. Referenced documents are listed below. When any of the following standards are superseded by an approved revision, the revision shall apply.

1. *Power ISA*
2. *IEEE 1275, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices*

   IEEE part number DS02683, ISBN 1-55937-426-8
3. *Core Errata, IEEE P1275.7/D4*
4. *Open Firmware Recommended Practice:OBP-TFTP extension*
5. *Open Firmware Recommended Practice: Device Support Extensions*
6. *PCI Bus binding to: IEEE Std 1275-1994, Standard for Boot (Initialization, Configuration) Firmware*
7. *Open Firmware: Recommended Practice - Interrupt Mapping*
8. *Open Firmware: Recommended Practice - Forth Source and FCode Image Support*, Version 1.0
9. *Open Firmware: Recommended Practice - Interrupt Mapping*, Version 1.0
10. *Open Firmware: Recommended Practice - TFTP Booting Extensions*, Version 0.8
11. *Open Firmware: Recommended Practice - Interposition*, Version 0.2
12. *MS-DOS Programmer's Reference*

    Published by Microsoft
13. *Peering Inside the PE: A Tour of the Win32 Portable Executable File Format*

    Found in the March, 1994 issue of *Microsoft Systems Journal*
14. *ISO-9660, Information processing -- Volume and file structure of CD-ROM for information interchange*

    Published by International Organization for Standardization
15. *System V Application Binary Interface, PowerPC Processor Supplement*

    By Sunsoft
16. *ISO Standard 8879:1986, Information Processing -- Text and Office Systems -- Standard Generalized Markup Language (SGML)*
17. *IEEE 996, A Standard for an Extended Personal Computer Back Plane Bus*
18. *PCI Local Bus Specification*

All designers are responsible for assuring that they use the most current version of this document at the time that they design conventional PCI related components or platforms. See the PCI SIG website for the most current version of this document.

19. *PCI-to-PCI Bridge Architecture Specification*

    All designers are responsible for assuring that they use the most current version of this document at the time that they design conventional PCI related components or platforms. See the PCI SIG website for the most current version of this document.

20. *PCI Standard Hot-Plug Controller and Subsystem Specification*

21. *PCI-X Protocol Addendum to the PCI Local Bus Specification*

    All designers are responsible for assuring that they use the most current version of this document at the time that they design PCI-X related components or platforms. See the PCI SIG website for the most current version of this document.

22. *PCI Express Base Specification*

    All designers are responsible for assuring that they use the most current version of this document at the time that they design PCI Express related components or platforms. See the PCI SIG website for the most current version of this document.

23. *PCI Express to PCI/PCI-X Bridge Specification*

    All designers are responsible for assuring that they use the most current version of this document at the time that they design PCI Express related components or platforms. See the PCI SIG website for the most current version of this document.

24. *System Management BIOS (SMBIOS) Reference Specification*

25. *(List Number Reserved for Compatibility)*

26. *(List Number Reserved for Compatibility)*

27. *(List Number Reserved for Compatibility)*

28. *IBM RS/6000® Division, Product Topology Data System, Product Development Guide*

    Version 2.1

29. *Single Root I/O Virtualization and Sharing Specification*

    All designers are responsible for assuring that they use the most current version of this document at the time that they design PCI Express SR-IOV related components or platforms. See the PCI SIG website for the most current version of this document.

30. *Multi-Root I/O Virtualization and Sharing Specification*

    All designers are responsible for assuring that they use the most current version of this document at the time that they design PCI Express MR-IOV related components or platforms. See the PCI SIG website for the most current version of this document.

Glossary

This glossary contains an alphabetical list of terms, phrases, and abbreviations used in this document.

Term	Definition
AC	Alternating current
ACR	Architecture Change Request
AD	Address Data line
Adapter	A device which attaches a device to a bus or which converts one bus to another; for example, an I/O Adapter (IOA), a PCI Host Bridge (PHB), or a NUMA fabric attachment device.
addr	Address
Architecture	The hardware/software interface definition or software module to software module interface definition.
ASCII	American National Standards Code for Information Interchange
ASR	Address Space Register
BAT	Block Address Translation
BE	Big-Endian or Branch Trace Enable bit in the MSR (MSR_{BE})
BIO	Bottom of Peripheral Input/Output Space
BIOS	Basic Input/Output system
BIST	Built in Self Test
Boundedly undefined	Describes some addresses and registers which when referenced provide one of a small set of predefined results.
BPA	Bulk Power Assembly. Refers to components used for power distribution from a central point in the rack.
BPM	Bottom of Peripheral Memory
BSCA	Bottom of System Control Area
BSM	Bottom of System Memory
BUID	Bus Unit Identifier. The high-order part of an interrupt source number which is used for hardware routing purposes by the platform.
CCIN	Custom Card Identification Number
CD-ROM	Compact Disk Read-Only Memory
CIS	Client Interface Service
CMO	Cooperative Memory Over-commitment option. See Section 14.12.3, “Cooperative Memory Over-commitment Option (CMO),” on page 474
CMOS	Complimentary Metal Oxide Semiconductor

Conventional PCI	Behavior or features that conform to *PCI Local Bus Specification* [18].
CPU	Central Processing Unit
CR	Condition Register
CTR	Count Register
DABR	Data Address Breakpoint Register
DAR	Data Address Register
DASD	Direct Access Storage Device (a synonym for "hard disk")
DBAT	Data Block Address Translation
DC	Direct current
DEC	Decrementer
DIMM	Dual In-line Memory Module
DMA	Direct Memory Access
DMA Read	A data transfer from System Memory to I/O. A DMA Read Request is the inbound operation and the DMA Read Reply (or Read Completion) is the outbound data coming back from a DMA Read Request.
DMA Write	A data transfer to System Memory from I/O or a Message Signalled Interrupt (MSI) DMA Write. This is an inbound operation.
DOS	Disk OS
DR	Data Relocate bit in MSR (MSR_{DR})
DRA	Deviation Risk Assessment
DRAM	Dynamic Random Access Memory
DRC	Delayed Read Completion. A transaction that has completed on the destination bus and is now moving toward the originating bus to complete. DR Connector.
DR entity	An entity that can participate in DR operations. That is, an entity that can be added or removed from the platform while the platform power is on and the system remains operational.
DRR	Delayed Read Request. A transaction that must complete on the destination bus before completing on the originating bus.
DSISR	Data Storage Interrupt Status Register
DWR	Delayed Write Request. A transaction that must complete on the destination bus before completing on the originating bus.
EA	Effective Address
EAR	External Access Register
ECC	Error Checking and Correction
EE	External interrupt Enable bit in the MSR (MSR_{EE})
EEH	Enhance I/O Error Handling
EEPROM	Electrically Erasable Programmable Read Only Memory

EPOW	Environment and Power Warning
Error Log indicator	An amber indicator that indicates that the user needs to look at the error log or problem determination procedures, in order to determine the cause. Previously called System Information (Attention).
FCode	A computer programming language defined by the OF standard which is semantically similar to the Forth programming language, but is encoded as a sequence of binary byte codes representing a defined set of Forth words.
FE0	Floating-point Exception mode 0 bit in the MSR (MSR_{FE0})
FE1	Floating-point Exception mode 1bit in the MSR (MSR_{FE1})
FIR	Fault Isolation Registers
FLR	Function Level Reset (see PCI Express documentation). An optional reset for PCI Express functions that allows resetting a single function of a multi-function IOA.
FP	Floating-Point available bit in the MSR (MSR_{FP})
FPSCR	Floating-Point Status And Control Register
FRU	Field Replaceable Unit
FSM	Finite State Machine
GB	Gigabytes - as used in this document it is 2 raised to the power of 30
HB	Host Bridge
HMC	Hardware Management Console - used generically to refer to the system component that performs platform administration function where ever physically located. The HMC is outside of this architecture and may be implemented in multiple ways. Examples include: a special HMC applications in another system, an external appliance, or in an LPAR partition using the Virtual Management Channel (VMC) interface to the hypervisor.
Hz	Hertz
IBAT	Instruction block address translation
ID	Identification
IDE	Integrated Device Electronics
IDU	Interrupt Delivery Unit
IEEE	Institute of Electrical and Electronics Engineers
I^2C	Inter Integrated-circuit Communications
I/O	Input/Output
I/O bus master	Any entity other than a processor, cache, memory controller, or host bridge which supplies both address and data in write transactions or supplies the address and is the sink for the data in read transactions.
I/O device	Generally refers to any entity that is connected to an IOA (usually through a cable), but in some cases may refer to the IOA itself (that is, a device in the device tree that happens to be used for I/O operations).
I/O Drawer	An enclosure in a rack that holds at least one PHB and at least one IOA.
ILE	Interrupt Little-Endian bit in MSR (MSR_{ILE})
Instr	Instruction

Interrupt Number	See Interrupt Vector below.
Interrupt Vector	The identifier associated with a specific interrupt source. The identifier's value is loaded into the source's Interrupt Vector Register and is read from the Interrupt Delivery Unit's Interrupt Acknowledge Register.
IOA	I/O Adapter. A device which attaches to a physical bus which is capable of supporting I/O (a physical IOA) or logical bus (a virtual IOA). The term "IOA" without the usage of the qualifier "physical" or "virtual" will be used to designate a physical IOA. Virtual IOAs are defined further in Chapter 17, "Virtualized Input/Output," on page 597. In PCI terms, an IOA may be defined by a unique combination of its assigned bus number and device number, but not necessarily including its function number. That is, an IOA may be a single or multi-function device, unless otherwise specified by the context of the text. In the context of a PCIe I/O Virtualized (IOV) device (not to be confused with a virtual IOA), an IOA is a single or multiple function device (for example, a PCIe Virtual Function (VF) or multiple VFs). An IOA function may or may not have its own set of resources, that is may or may not be in its own Partitionable Endpoint (PE) domain (see also Section 4.1, "I/O Topologies and Endpoint Partitioning," on page 71).
IOA function	That part of an IOA that deals with a specific part of the IOA as defined by the configuration space "Function" part of Bus/Device/Function. For single-function IOAs, the IOA Function and the IOA are synonymous.
IP	Interrupt Prefix bit in MSR (MSR_{IP})
IPI	Interprocessor Interrupt
IR	Instruction Relocate bit in MSR register (MSR_{IR}) or infrared
ISF	Interrupt 64-bit processor mode bit in the MSR (MSR_{ISF})
ISO	International Standards Organization
ISR	Interrupt Source Register
ISU	Interrupt Source Unit
KB	Kilobytes - as used in this document it is 2 raised to the power of 10
KHz	Kilo Hertz
LAN	Local Area Network
LCD	Liquid Crystal Display
LE	Little-Endian bit in MSR (MSR_{LE}) or Little-Endian
LED	Light Emitting Diode
LMB	Logical Memory Block. The Block of logical memory addresses associated with a dynamically reconfigurable memory node.
Load	A *Load* Request is the outbound (from the processor) operation and the *Load* Reply is the inbound data coming back from a *Load* Request. When it relates to I/O operations, this is an MMIO *Load*.
LR	Link Register
LSb	Least Significant bit
LSB	Least Significant Byte
LSI	Level Sensitive Interrupt

LUN	Logical Unit Number
L1	Primary cache
L2	Secondary cache
MB	Megabytes - as used in this document it is 2 raised to the power of 20
ME	Machine check Enable
MMIO	Memory Mapped I/O. This refers to the mapping of the address space required by an I/O device for *Load* or *Store* operations into the system's address space.
MES	Miscellaneous Equipment Specification
MFM	Modified frequency modulation
MHz	Mega Hertz
MOD	Address modification bit in the MSR (MSR_{MOD})
MP	Multiprocessor
MSb	Most Significant bit
MSB	Most Significant Byte
MSI	Message Signalled Interrupt
MSR	Machine State Register
MTT	Multi-TCE-Table option. See Section 14.5.4.2.4, "H_PUT_TCE_INDIRECT," on page 421.
N/A	Not Applicable
Nibble	Refers to the first or last four bits in an 8 bit byte
NUMA	Non-Uniform Memory Access
NUMA fabric	Mechanism and method for connecting the multiple nodes of a NUMA system
NVRAM	Nonvolatile Random Access Memory
OF	Open Firmware
OP	Operator
OS	Operating System
OUI	Organizationally Unique Identifier
PA	Processor Architecture
PAP	Privileged Access Password
LoPAPR	Used within the Linux on Power Architecture Platform Reference document to denote: (1) the architectural requirements specified by the Linux on Power Architecture Platform Reference document, (2) the Linux on Power Architecture Platform Reference document itself, and (3) as an adjective to qualify an entity as being related to this architecture.
Partitionable Endpoint	This refers to the I/O granule that may be treated as one for purposes of assignment to an OS (for example, to an LPAR partition). May be an I/O adapter (IOA), or groups of IOAs and bridges, or portions of IOAs. PE granularity supported by the hardware may be finer than is supported by the firmware. Grouping of multiple PEs into one DR entity may limit assignment of a the separate PEs to different LPAR partitions. See also DR entity.
PC	Personal Computer

PCI	Peripheral Component Interconnect. An all-encompassing term referring to conventional PCI, PCI-X, and PCI Express.
PCI bus	A general term referring to either the PCI Local Bus, as specified in *PCI Local Bus Specification* [18] and *PCI-X Protocol Addendum to the PCI Local Bus Specification* [21] for conventional PCI and PCI-X, or a PCI Express link, as specified in *PCI Express Base Specification* [22] for PCI Express.
PCI Express	Behavior or features that conform to *PCI Express Base Specification* [22].
PCI link	A PCI Express link, as specified in *PCI Express Base Specification* [22].
PCI-X	Behavior or features that conform to *PCI-X Protocol Addendum to the PCI Local Bus Specification* [21].
PD	Presence Detect
PE	When referring to the body of the LoPAPR, this refers to a Partitionable Endpoint. PE has a different meaning relative to Appendix B, "LoPAPR Binding," on page 661 (see Section B.3, "Terms," on page 661 for that definition).
PEM	Partition Energy Management option. See Section 14.14, "Partition Energy Management Option (PEM)," on page 492.
Peripheral I/O Space	The range of real addresses which are assigned to the I/O Space of a Host Bridge (HB) and which are sufficient to contain all of the *Load* and *Store* address space requirements of all the devices in the I/O Space of the I/O bus that is generated by the HB. A keyboard controller is an example of a device which may require Peripheral I/O Space addresses.
Peripheral Memory Space	The range of real addresses which are assigned to the Memory Space of a Host Bridge (HB) and which are sufficient to contain all of the *Load* and *Store* address space requirements of the devices in the Memory Space of the I/O bus that is generated by the HB. The frame buffer of a graphics adapter is an example of a device which may require Peripheral Memory Space addresses.
Peripheral Space	Refers to the physical address space which may be accessed by a processor, but which is controlled by a host bridge. At least one peripheral space must be present and it is referred to by the suffix 0. A host bridge will typically provide access to at least a memory space and possibly to an I/O space.
PHB	PCI Host Bridge
PIC	Programmable Interrupt Controller
PIR	Processor Identification Register
Platform	Refers to the hardware plus firmware portion of a system composed of hardware, firmware, and OS.
Platform firmware	Refers to all firmware on a system including the software or firmware in a support processor.
Plug-in I/O card	A card which can be plugged into an I/O connector in a platform and which contains one or more IOAs and potentially one or more I/O bridges or switches.
Plug-in Card	An entity that plugs into a physical slot.
PMW	Posted memory write. A transaction that has complete on the originating bus before completing on the destination bus
PnP	Plug and Play
POP	Power On Password

POST	Power-On Self Test
PR	Privileged bit in the MSR (MSR_{PR})
Processor Architecture	Used throughout this document to mean compliance with the requirements specified in *Power ISA* [1].
Processor revision number	A 16-bit number that distinguishes between various releases of a particular processor version, for example different engineering change levels.
PVN	Processor Version Number. Uniquely determines the particular processor and PA version.
PVR	Processor Version Register. A register in each processor that identifies its type. The contents of the PVR include the processor version number and processor revision number.
RAID	Redundant Array of Independent Disks
RAM	Random Access Memory
RAS	Reliability, Availability, and Serviceability
Real address	A real address results from doing address translation on an effective address when address translation is enabled. If address translation is not enabled, the real address is the same as the effective address. An attempt to fetch from, load from, or store to a real address that is not physically present in the machine may result in a machine check interrupt.
Reserved	The term "reserved" is used within this document to refer to bits in registers or areas in the address space which should not be referenced by software except as described in this document.
Reserved for firmware use	Refers to a given location or bit which may not be used by software, but are used by firmware.
Reserved for future use	Refers to areas of address space or bits in registers which may be used by future versions of this architecture.
RI	Recoverable interrupt bit in the MSR (MSR_{RI})
RISC	Reduced Instruction Set Computing
RMA	Real Mode Area. The first block of logical memory addresses owned by a logical partition, containing the storage that may be accessed with translate off.
ROM	Read Only Memory
Root Complex	A PCI Express root complex as specified in *PCI Express Base Specification* [22].
RPN	Real Page Number
RTAS	Run-Time Abstraction Services
RTC	Real Time Clock
SAE Log	Service Action Event log
SCC	Serial Communications Controller
SCSI	Small Computer System Interface
SE	Single-step trace enabled bit in the MSR (MSR_{SE})
Service Focal Point	The common point of control in the system for handling all service actions
Serviceable Event	Serviceable Events are platform, global, regional and local error events that require a service action and possibly a call home when the serviceable event must be handled by a service representative or at least reported to the service provider. Activation of the Error Log indicator noti-

	fies the customer of the event and the event indicates to the customer that there must be some intervention to rectify the problem. The intervention may be a service action that the customer can perform or it may require a service provider.
SES	Storage Enclosure Services (can also mean SCSI Enclosure Services in relation to SCSI storage)
SF	Processor 32-bit or 64-bit processor mode bit in the MSR (MSR_{SF})
SFP	Service Focal Point
Shrink-wrap OS	A single version of an OS that runs on all compliant platforms.
Shrink-wrap Application	A single version of an application program that runs on all compliant platforms with the applicable OS.
SMP	Symmetric multiprocessor
SMP Mode	Non-LPAR mode. That is, when there is no hypervisor running.
SMS	System Management Services
Snarf	An industry colloquialism for cache-to-cache transfer. A typical scenario is as follows: (1) cache miss from cache A, (2) line found modified in cache B, (3) cache B performs castout of modified line, and (4) cache A allocates the modified line as it is being written back to memory.
Snoop	The act of interrogating a cache for the presence of a line, usually in response to another party on a shared bus attempting to allocate that line.
SPRG	Special Purpose Registers for General use
SR	System Registers
SRC	Service Reference Code
SRN	Service Request Number
Store	A *Store* Request is an outbound (from the processor) operation. When it relates to I/O operations, this is an MMIO *Store*.
System	Refers to the collection of hardware, system firmware, and OS software which comprise a computer model.
System address space	The total range of addressability as established by the processor implementation.
System Control Area	Refers to a range of addresses which contains the system ROM(s) and an unarchitected, reserved, platform-dependent area used by firmware and Run-Time Abstraction services for control of the platform. The ROM areas are defined by the OF properties in the **`openprom`** and **`os-rom`** nodes of the OF device tree.
System Information (Attention) indicator	See Error Log indicator.
System firmware	Refers to the collection of all firmware on a system including OF, RTAS and any legacy firmware.
System Memory	Refers to those areas of memory which form a coherency domain with respect to the PA processor or processors that execute application software on a system.
System software	Refers to the combination of OS software, device driver software, and any hardware abstraction software, but excludes the application software.
TB	Time Base

TCE	Translation Control Entry
TLB	Translation Look-aside Buffer
TOD	Time Of Day
TOSM	Top of system memory
TPM	Top of Peripheral Memory
tty	Teletypewriter or ASCII character driven terminal device
UI	User Interface
USB	Universal Serial Bus
v	Volt
VGA	Video Graphics Array
VMC	Virtual Management Channel
VPD	Vital Product Data
VPNH	Virtual Processor Home Node option. See Section 14.11.6, “Virtual Processor Home Node Option (VPHN),” on page 467.

End of LoPAPR Document