AXIe-2: Base Software Specification

Revision 2.2
Including provisional changes for revision 2 of AXIe-1

September 13, 2018

Important Information

Notice

AXIe-2: Base Software Specification is authored by the AXIe Consortium companies. For a vendor membership roster list, please contact execdir@axiestandard.org.

The AXIe Consortium wants to receive your comments on this specification. To provide such feedback, or to join the consortium, contact execdir@axiestandard.org.

Warranty

The AXIe Consortium and its member companies make no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Its member companies shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Trademarks

PICMG, CompactPCI, and AdvancedTCA® are registered trademarks of the PCI Industrial Computers Manufacturers Group.

PCI-SIG, PCI Express, and PCIe are registered trademarks of PCI-SIG.

LXI is a trademark of the LXI Consortium Inc.

PXI is a trademark of the PXI Systems Alliance.

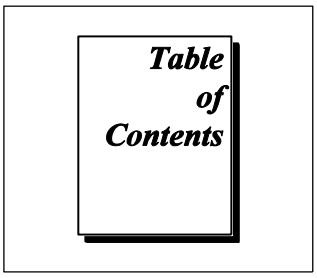
IVI is a trademark of the IVI Consortium

VXI*plug&play* is a trademark of the IVI Consortium

VISA is a trademark of the IVI Consortium

Other product and company names listed are trademarks or trade names of their respective companies.

No investigation has been made of common-law trademark rights in any work.



Revision History

This section is an overview of the revision history of the AXIe-2.0 software specification.

Table 1-1. Software Specification Revisions

Revision Number	Date of Revision	Revision Notes
1.0	November 21, 2011	Initial revision of the spec
1.0	January 6, 2012	Clarified how the system module should respond to mimic a Compact PCI system module.
2.0	March, 2016	Various changes: • Updates to reflect that slot 2 is no longer a hub slot. • Updates to reflect 4 4-lane links to each peripheral slot • Deleted FRUIndex in GetFieldCount in section 4.2.4(6.2.4 per 2.2 changes) • Clarified Rule 3.1 to specify requirements with respect to AXIe 1.0 Rev 3. • Updated references to the AXIe specifications to the new format. E.g., AXIe-1 instead of AXIe 1.0. • Updated 1.9 and 1.12 to reflect updates to IVI and PXI standards. Provisional Changes: • Added rule 3.13 in section 3.4.3 regarding how to report system slot link width.(5.13 per 2.2 changes) • Modified Rule 4.2 in section 4.2 to require AXIe extensions intead of them being optional. (6.2 per 2.2 changes) • Provisional additions in 4.2.6 and 4.2.7 to support reporting AXIe 1.0 Rev 3 fabric information(6.2.6 and 6.2.7 per 2.2 changes)
2.1	January 11, 2018	Added section 1.1 to clarify compliance requirements, and clarified revision as 2.1, which includes provisional changes.
2.2	August 1, 2018	Updates to align with reent PXI revisions. These add: • Support for Linux operating systems • Support for modules that occupy multiple slots

Table 1-1. Software Specification Revisions

		• Support for modules that have multiple endpoints in the same or different slots. • Updated section 1.8 to permit devices to cite compliance to this version of the spec and permitting additional compliance statements if they also implement the provisional changes.

1 Introduction

1.1 AXIe Compliance

This specification is for use with other AXIe specifications. The AXIe consortium places limitations on the use of the AXIe trademark and requires compliance with certain companion standards. Detailed compliance requirements and trademark usage requirements are in the AXIe Compliance Requirements and Trademark Usage document.

1.2 Overview of the AXIe-2: Base Software Specification

AXIe-2 defines software specifications that support the AXIe 1.0 Base Architecture. Both architectures define an extensible platform for general purpose, modular instrumentation. The architecture incorporates the best features of earlier modular open instrumentation platforms, including VXI, PXI, and LXI. Like VXI and PXI, the architecture is based on a popular modular computing platform with added features important to developers and users of test and measurement systems.

The base platform is AdvancedTCA®, an open architecture for modular computing components targeted for communications infrastructure applications. The AdvancedTCA® architecture includes provisions for power distribution, power and system management, Ethernet communication between modules, and PCI Express communication between modules, along with other capabilities.

The AXIe 1.0 Base Architecture includes some modifications to the AdvancedTCA® architecture. These modifications provide timing, triggering, and module-to-module data movement features that are important to the implementation of high-performance test and measurement systems. These modifications are designed to allow most AdvancedTCA® computing modules to work within an AXIe 1.0 environment, to allow many general-purpose AXIe devices to work in AdvancedTCA® environments, and to prevent any damaging incompatibilities between AXIe and AdvancedTCA® devices and system components.

1.3 Objectives

The AXIe-2 Base Software Specification specifies the software necessary to support AXIe chassis, system modules, and instrument modules that comply with AXIe-1 Base Hardware Architecture Specification.

The AXIe-2: Base Software Specification builds on the PXI software specifications in much the same way that the AXIe-1: Base Hardware Specification builds on the AdvancedTCA® specifications. Specifically, AXIe software is required to substantially comply with PXI-2, *the PXI Software Specification* and PXI-6, the *PXI Express Software Specification*. This permits AXIe system software to be supported by existing PXI utilities; it also facilitates integrating systems composed of both PXI and AXIe components.

1.4 Audience

This specification is intended to be used by:

- Product developers interested in implementing and utilizing software features of the AXIe platform.
- Hardware designers interested in referencing description files that identify and describe AXIe hardware capabilities.
- Software designers interested in utilizing descriptors for managing AXIe resources, such as AXIe triggers, local busses, and geographic parameters for slot and chassis identification.

1.5 References

Several other documents and specification are related to this specification. These other related documents are the following:

VPP-4.3 Defines the semantics of the VISA library (see www.ivifoundation.org)

VPP-4.3.2 VISA Implementation Specification for Textual Languages (see: www.ivifoundation.org)

IVI-6.3 VISA PXI Plug-in Specification (see: www.ivifoundation.org)

PXI-2 PXI Software Specification (see: www.pxisa.org). References to PXI-2 in this document are to PXI-2 Revision 2.5, May 31, 2018.

PXI-4 PXI Module Description File Specification (see: www.pxisa.org). References to PXI-4 in this document are to PXI-4 Revision 1.2, May 31, 2018

PXI-6 PXI Express Software Specification (see: www.pxisa.org). References to PXI-6 in this document are to PXI-6 Revision 1.3, May 31, 2018.

PXI-8 PXI MultiComputing Software Specification (see: www.pxisa.org). References to PXI-8 in this document are to PXI-8 Revision 1.1, May 31, 2018.

CompactPCI PICMG 2.0 Specification

1.6 Definitions of Terms and Acronyms

This section defines terms and acronyms used in this specification:

PXI This term originates from PXISA.org and refers broadly to the PXI specifications and compliant components; this includes both PXI Express and PXI-1.

PXI-1 This term originates from PXISA.org and refers specifically to PXI modules that are based on parallel PCI and not PCI Express.

System A system is a collection of one or more chassis. Each chassis may be an AXIe chassis or a PXI chassis.

Chassis A chassis is a unit that contains a backplane, a system module, instrument modules, and a shelf manager.

Instrument Module This is a card that plugs into an instrument slot within an AXIe chassis. In AdvancedTCA®, these cards are loosely referred to as “blades”.

Peripheral Module This term is synonymous with Instrument Module. It is commonly used in PXI and Compact PCI. This term is generally used in this document to be consistent with the referenced PXI specifications.

System Module The system module provides the communication interface, both network and PCIe, outside the chassis. The system module also distributes these communication fabrics among the modules.

Shelf Manager The Shelf Manager Module, also referred to as the *ShMM*, interacts with modules in the chassis as well as other Field Replaceable Units (FRU), such as power supplies and fans. The shelf manager maintains an inventory of these units and is responsible for powering and monitoring of modules over an independent IPMB backplane bus. Much more information may be found in PICMG 3.0.

System Module Driver The driver defined by PXI-6 that enumerates system modules and the chassis they connect to and provides API access to the chassis and system modules.

Chassis Driver The driver defined by PXI-6 that returns information about the chassis. In AXIe and PXI Express chassis information is primarily provided by the system module driver, however AXIe-2 defines additional APIs that provide FRU information.

Peripheral Module Driver The driver defined by PXI-6 that returns information about its supported peripheral modules.

IPMC Intelligent Platform Management Controller. The microcontroller that provides a IPMB connection to the ShMC. Although required, on an instrument module, an IPMC may reside in the chassis and provide other services such as power supply monitoring.

ShMC Shelf Manager Controller. The microcontroller that resides on the shelf and is the master of the IPMB.

FRU Field replaceable unit. This term refers to a set of functionality on an AXIe module. This is a historical term and the referenced functionality may not actually be a replaceable unit.

1.7 Organization

This specification was created and is maintained under the rules of the AXIe consortium.

This specification consists of a system of numbered RULES, RECOMMENDATIONS, PERMISSIONS, and OBSERVATIONS, along with supporting text, tables, and figures.

RULEs outline the requirements of the specification. They are characterized by the keyword "**SHALL**". Conformance to these rules provides the necessary level of compatibility to support the multi-vendor interoperability expected by system integrators and end users in the test and measurement industry. Products that conform to this specification must meet all of the requirements spelled out in the various rules.

RECOMMENDATIONs provide additional guidance that will help AXIe equipment manufacturers improve their users' experiences with AXIe systems. They are characterized by the keyword "**SHOULD**". Following the recommendations should improve the functionality, flexibility, interoperability, and/or usability of AXIe products. Products are not required to implement the recommendations.

PERMISSIONs explicitly highlight some of the flexibility of the AXIe specification. They are characterized by the keyword "**MAY**". The permissions generally clarify the range of design choices that are available to product and system designers at their discretion. They allow designers to trade off functionality, cost, and other factors in order to produce products that meet their users' expectations. Permissions are neutral and imply no preference as to their implementation.

OBSERVATIONs explicitly highlight some of the important nuances of the specification. They help the readers to fully understand some of the implications of the specification's requirements and/or the rationale behind particular requirements. They generally provide valuable design guidance.

All rules, recommendations, permissions, and observations must be considered in the context of the surrounding text, tables, and figures. Rules may explicitly or implicitly incorporate information from the text, tables, and figures. Although the authors of this specification have gone to significant effort to insure that all of the necessary requirements are spelled out in the rules, it is possible that some important requirements appear only in the specification's free text. Conservative design practice dictates that such embedded requirements be treated as rules.

The AXIe-2 specification is based on several PXISA® specifications and AXIe 1.0 hardware specifications. Relevant PXISA® numbered requirements are explicitly referenced in this specification's rules, recommendations, permissions, and observations. These requirements are incorporated along with their supporting context (text, tables, figures, etc.). Any numbered requirement that is not explicitly included by this specification's rules, recommendations, permissions, or observations, is excluded from the AXIe-2 requirements.

Successful implementation of AXIe products and systems requires in-depth knowledge of the AXIe 1.0 Base Architecture Specification and PXISA® specifications.

1.8 Provisional Changes for Provisional Revision P2.0 of AXIe-2

The initial release of revision 2.0 of this specification was passed provisionally. That is, the AXIe consortium believes the changes made for revision 2.0 to be accurate and complete further the consortium expects them to be required for compliance with the specification. However, subject to implementation and interoperation testing, the changes and additions to the specification may need to be revisited. Once suitably verified, the changes will be either confirmed or, if necessary, extension and corrections consistent with the direction established by the provisional changes may be added.

The provisional changes are:

- Added Rule 5.14 in section 5.4.3 regarding how to report system slot link width.
- Modified Rule 6.2 in section 6.2 to require AXIe extensions intead of them being optional.
- Provisional additions in 6.2.6 and 6.2.7 to support reporting AXIe 1.0 Rev 3 fabric information

Any implementation the complies with all of the non-provisional rules is permitted to cite compliance to AXIe-2, Revision 2.2. Implementations that further comply with the provisional rules listed above are permitted to cite compliance to "AXIe-2 Revision 2.2, provisional changes".

1.9 Extensions to the AXIe-2: Base Software Specification

The AXIe modular architecture is intended to be an extensible architecture. An extension may focus on a specific niche market requirement that is not considered advantageous to be a part of the base specification. In contrast the base specification describes attributes and features that are advantageous to be common across all extensions. Such commonality is beneficial across the AXIe community.

An example of a hardware extension might be a specific market application using the Zone 3 interface. Such an extension likely has software features targeted for using that hardware extension. These hardware and software extensions would be described in separate hardware and software specifications specific to the extension. Such extensions must be adopted and approved by the AXIe Consortium Committee.

1.10 Use of the PXI Software Specifications

This specification leverages from well known and understood standards such as the PXI specifications. Where possible, the AXIe standard directly uses the PXI software specification. This simplifies implementation, system integration, promotes adoption, and encourages reuse.

In addition, AXIe contains elements that do not exist in PXI but must be supported by the base software. The requirements in this specification regarding those elements do not conflict with any PXI requirements.

No provisions in this specification conflict with, or alter the PXI requirements.

The PXISA has defined a mechanism for software clients to reserve backplane trigger lines. AXIe chassis are encouraged to comply with PXI-9 in order for clients to reserve AXIe trigger lines (note that routing is not meaningful for AXIe chassis).

The PXI-8, *PXI MultiComputing Software Specification* defines shared memory and other standard protocols to allow CPU based device to communicate through non-transparent bridges. Compliance with PXI-8 is optional for AXIe equipment.

1.11 Use of the LXI Software Specification

The AXIe software specification recommends that AXIe network modules comply with LXI where practical. Further details of software requirements for the network connection on modules may be incorporated in a latter version of this standard.

The shelf manager in an AXIe system has a network interface. In AdvancedTCA® systems, the shelf manager is statically configured to a fixed IP address during configuration. AXIe systems are more dynamic in that they may be set up, configured, modified, and moved to suit a particular test configuration. The network fabric in these cases may be part of a DHCP served environment in which IP addresses are subject to change. Chassis and system designers should create their shelf managers to be friendly entities on the network and attain IP addresses using DHCP or Dynamically Configured Link Local Addressing[1].

1.12 Use of AdvancedTCA® Software Specifications

AdvancedTCA® provides a rich feature set of hardware and software functionality. The AXIe 1.0 Base Architecture Specification, Chapter 3, *Hardware Platform Management*, summarizes important attributes of AdvancedTCA® and how these relate to an AXIe system context. This chapter references several other specifications that describe e-keying functionality, FRU records, the Intelligent Platform Management Interface (IPMI 1.5), and Remote Management Control Protocol (RMCP). These standards permit a common set of software tools to interact with shelf and board management features. There are vendors today that already specialize in promoting shelf managers and board management controllers adhering to these AdvancedTCA® recommendations.

1.13 VISA Specification

The IVI Foundation defines a family of VXI*plug&play* specifications that include the VISA specifications which define a standard IO library. The basic capabilities of the library are defined in the VPP-4.3 specification, *The VISA Library*. VPP-4.3.2, *VISA Implementation Specification for Textual Languages*, defines how the library is implemented for textual languages.

The IVI suite of VISA standards includes IVI-6.3, *IVI VISA PXI Plug-in* which defines a stand-alone VISA plug-in which may be called by VISA to perform PXI IO for a specific device. Although beyond the scope of this specification, this is a useful mechanism for doing IO to AXIe devices that comply with this specification.

This is especially useful since PXI-2 section 3.6 requires that all system controllers shall provide a VISA implementation that supports the PXI bus and complies with the VISA library specifications, version 4.0 or higher. By using the VISA Plug-In, vendors need not provide a full VISA library.

Information regarding the VISA standards is available from the IVI Foundation.

[1] Per RFC 3927, *Dynamic Configuration of Ipv4 Link-Local Addresses*

2 AXIe Software Architecture Overview

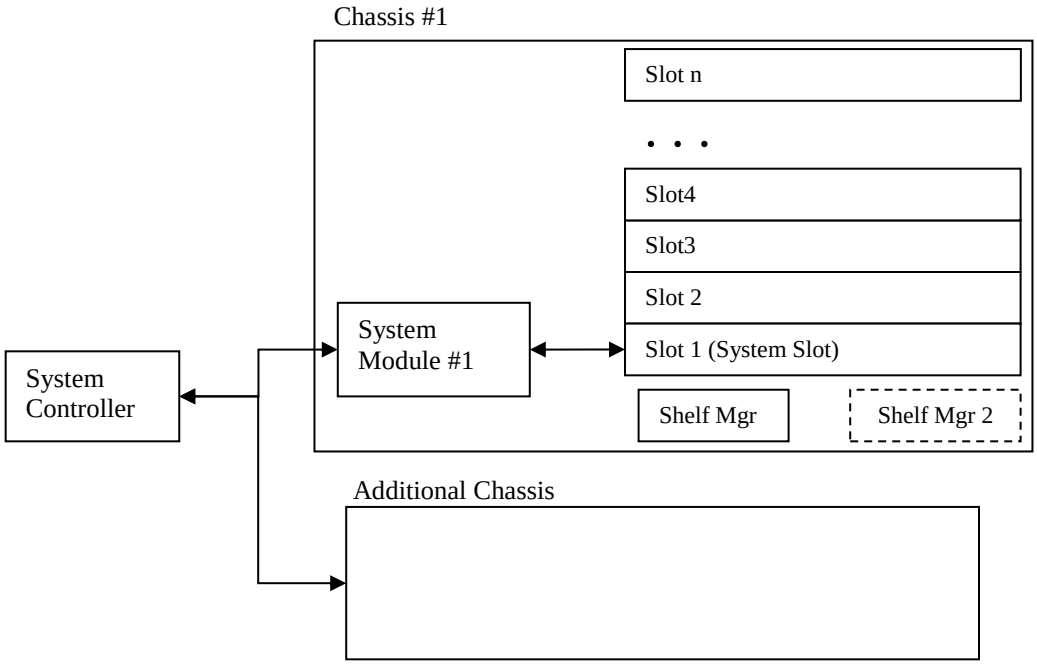


Figure 1 Hardware Components of an AXIe System

An AXIe system's heritage is based upon AdvancedTCA®, PICMG 3.0, and PXI. The AXIe architecture adds several extensions beyond the scope of telecommunication applications into test and measurement applications. AXIe provides both network and PCIe interconnect fabrics. Additionally, there are dedicated clocks, synchronization, and triggering signals specifically added for instrumentation. For more details on the hardware architecture, refer to the AXIe 1.0 Base Architecture Specification.

The key components of the system are shown in Figure 1, *Hardware Components of an AXIe System*. Note that:

- There is a single host computer in the system, called the *system controller*. Multiple system modules may be connected to the system controller. Although the host computer may be embedded in the AXIe chassis, the software architecture remains the same.

- The *system module* (SM) is the hardware that provides the connectivity to the chassis. The system module must occupy the system slot in an AXIe chassis, and provide connectivity to the system controller. There is one system module per chassis.

- AXIe-2 uses the physical slot number to reference slots. Throughout this document, any reference to slots refers to the physical slot unless further qualified. The physical slot is the slot designation that is visible to a user looking at the front of the chassis, and are labelled from left to right or bottom to top per ATCA specifications. Although individual modules and their software may not know the physical address, the chassis can provide that information using ATCA specified protocols.

  The physical slot is available to AXIe modules by interrogating the ShMC which maintains a site number lookup list which can be interrogated using standard ATCA netFN (net function) commands over IPMB.

- Each chassis contains a number of slots, including one system slot, an instrument hub slot, and instrument slots (also known as peripheral slots). The system slot is always logical address 1, but not necessarily physical slot 1. Furthermore, a chassis vendor may choose to integrate system module functionality into a chassis. In this case, the system module is referred to as a*n embedded system module* and the physical slot

number reported shall be 0. A chassis that has an embedded system module is referred to as a*n integrated chassis*.

- The shelf manager is associated with an individual chassis. Although not required as part of AXIe, there may be a redundant shelf manager for the purpose of high availability systems. This backup shelf manager transparently assumes the role of the primary shelf manager in the event of failure.

The AXIe hardware specification permits combining system controllers, system modules, chassis, and instrument modules from different vendors. The basic purpose of the AXIe-2 Base Software Specification is to provide the minimum software infrastructure to permit system providers to assemble systems from different components without tying the software to a specific vendor's implementation.

2.1 AXIe System Software Components

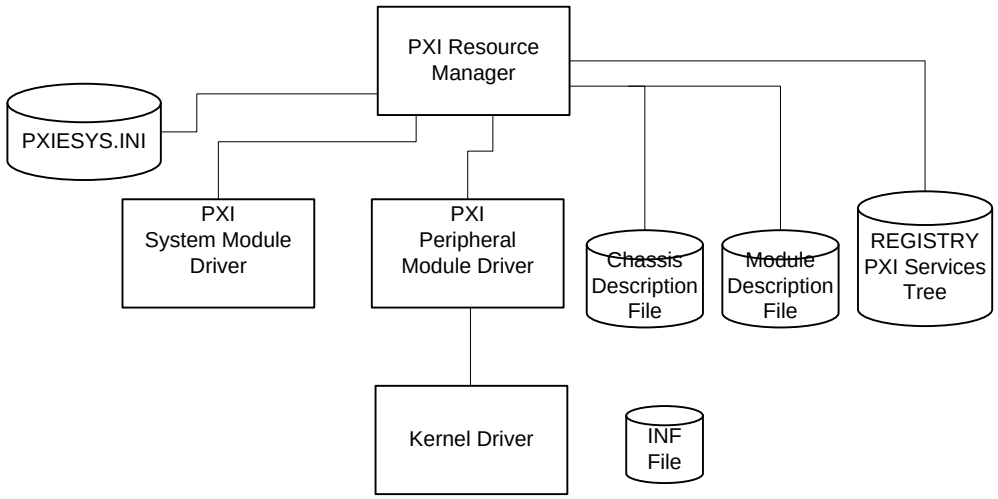


Figure 2 Basic AXIe system discovery showing the PXI modules used.

Figure 2 shows the PXI components that are involved in the discovery and enumeration of an AXIe system. The requirements regarding this software are the purview of the PXI specifications. An overview of their operation and how they work with AXIe is in Appendix A.

Details of this enumeration process are specified PXI-2 and PXI-6.

PXI-4 describes an optional module description file. This file contains information about the specific module. The resource manager may incorporate this information in the system description file (that is, the PXIESYS.INI file). In this case, The information from the module description file is embedded within the descriptor for the slot that contains this module in the system description file. For instance, if a module *foo* has a module description file with an entry *bar = "baz".* For each slot that contains the foo module, the PXIESYS.INI file will include the descriptor line *bar="baz".*

The module description file is optional for both PXI and AXIe devices.

The INF File shown in the illustration is provided along with the module kernel driver in order for the Windows operating system to correctly identify and install the kernel driver. Linux installations do not have this mechanism.

See Appendix A, Summary of the PXI Software Architecture, for additional detail on the PXI software architecture.

3 Multi-Slot, Multi-Link and Multi-Endpoint Modules

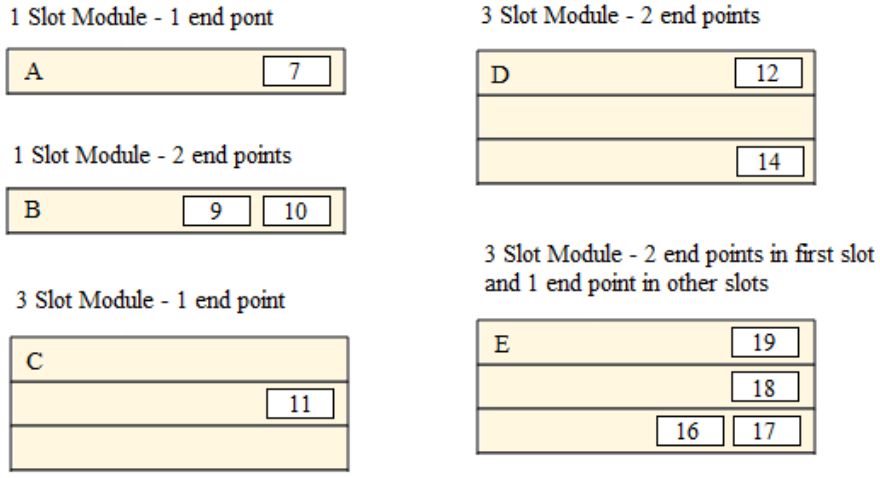


Figure 3 Examples of Multi-Slot, Multi-Link and Multi-Endpoint Modules

This topic describes the discovery mechanisms specified by PXI for:

- Modules that occupy multiple slots
- Modules that place multiple endpoints in a single slot

3.1 Multi-Slot Modules

Multi-Slot Modules occupy more than one slot. In Figure 3, C, D and E are Multi-Slot Modules. Effective with the PXI standard updates in 2018, the peripheral module driver supports calls that return:

- Slot Number – Slot number where the link is currently connected
- Occupied slot count – Number of slots occupied by this device
- Slot number offset – Number of chassis slots the module occupies that are to the left of the slot reported. For example, zero indicates that the occupied slot is the leftmost slot, and the module extends to the right by the occupied slot count minus 1.

3.2 Multi-Endpoint Modules

In Figure 3, B is an example of a multi-endpoint module. It contains a switch and two endpoints. Note that the operating system instantiates a kernel driver for each endpoint and each represents a PCI Express link. (Note, in PCI, the term link refers to a PCI connection between two devices).

PXI specifies that the API used to query links only report a single link for this slot, even though there are PCI Express links to both bus 9 and bus 10. PXI recommends that the connection to the backplane (from the switch) be reported from the module, but this is frequently not practical because there is not a generic way to determine the path to the switch on some operating systems. Therefore, peripheral module drivers are permitted to onlyreport the links to the endpoints.

To report two endpoints, the peripheral module driver returns the address of both endpoints in the address_info query as a semicolon separated string:

- For a single endpoint device, the address_info returns a VISA bus, device, function address string. For example: "PXI0::9-0.0::INSTR" (not including the quotes)

- For a multiple endpoint module, the peripheral module driver returns the VISA address for each endpoint, separated by commas. For example: "PXI0::9-0.0::INSTR;PXI0::10-0.0::INSTR" (not including the quotes) indicates two endpoints with bus addresses 9 and 10.

For instance, the first identified end-point address may be the main FPGA, the second endpoint could be for in-bound streaming, and the third for out-bound streaming. Thus, the instrument driver can access this string to determine how the module was enumerated.

The PXI standard does not specify the significance of the order of the addresses in the address_info string. However, the IVI VISA standard specifies that VISA sessions can be opened to each endpoint by specifying the index of the endpoint in the string.This is important because neither the PCI specification nor the operating system guarantee the order in which the PCI bus numbers are assigned for each endpoint on the module.

This same technique is used when a multi-endpoint module within a multi-slot module is reported.

3.3 Multi-Link Modules

The PXI Express standard refers to modules that occupy multiple slots where endpoints are connected to the backplane from more than one slot as multi-link modules. This is somewhat of a misnomer in that a multi-endpoint module also has multiple PCI links. These links all appear to be part of a single multi-slot module to the end user. In Figure 3, D and E are Multi-Link Modules.

For these modules, PXI specifies that a peripheral module driver reports the presence of a link in each slot that contains a link to the backplane. Therefore, the module reports both links, each describing the location and full width of the module and the offset of this link within the module. Thus, it is apparent to the resource manager that the module is three slots wide, and that there are links to the first and third slots because the installed peripheral module drivers report overlapping geographical locations for the modules.

4 Linux Support

PXI specifies Linux and Windows frameworks for both 32 and 64-bit operating systems. The architecture of the two is substantially the same, and thus the operating system has very little impact on the standard itself. The most notable difference is that the registration and configuration information on Windows is stored in the Windows 32-bit registry. On Linux, configuration information is stored in the file system.

AXIe-2 permits AXIe components to support any of the PXI-defined frameworks, including Linux, Windows, or both. Products should provide clear documentation of the supported framework.

In the case of Windows, the Windows registry provides a hierarchical organization for the installation data, and all installation data is stored in the 32-bit registry in a registry hive specified by the PXI standards alliance. Since Linux does not have an intrinsic data storage mechanism the configuration information is stored in a specified location on the Linux file system in INI files. The Linux file system hierarchy replicates the registry hierarchy on Windows.

PXI-2 specifies the location where the registration data is placed on the file system, and how the INI files are formatted and organized within the file system to provide the registration information required by PXI and AXIe. The PXI-2 specification also specifies the details of the group and user ownership of the created files.

INI files themselves do not have a tightly specified syntax, therefore the PXI standard specifies the syntax used in the INI files. This is a simple and conventional INI format and is the same format that is used for other INI files used in both the Windows and Linux frameworks. For instance the PXIESYS.INI file and the chassis INI file use the same syntax.

RULE 4.1: AXIe devices that support Linux shall comply with one or both of the Linux frameworks as specified by PXI in PXI-2 and PXI-6

RULE 4.2: AXIe devices that support Windows shall comply with one or both of the Windows frameworks as specified by PXI in PXI-2 and PXI-6.

5 AXIe Module Software Requirements

In order to facilitate the development and deployment of AXIe systems alongside PXI systems, the AXIe software requirements are based on the PXI software requirements.

RULE 5.1: AXIe chassis providers, system module providers, and instrument module providers are required to provide software components as required by the PXI standards for PXI Express devices. The relevant PXI standards for these devices are PXI-2, *PXI Software Specification*, and PXI-6, *The PXI Express Software Specification*.

In addition, AXIe providers are required to provide software that complies with all other rules in this specification. AXIe hardware that complies with earlier versions of AXIe-1 than 3.0 shall comply with AXIe-2 revision 1.0. AXIe hardware that complies with AXIe-1 version 3.0 shall comply with AXIe-2 revision 2.0 or later.

Since there are certain intrinsic differences between PXI and AXIe, a small number of differences exist between the AXIe deliverables and PXI deliverables. In some cases, the differences are only in the implementation, in other cases, the PXI deliverable may not be relevant for AXIe.

Further, since AXIe provides capabilities beyond the scope of PXI, these are described separately in a way that permits compatibly deploying AXIe and PXI, and also provides the necessary incremental information for AXIe peripherals.

5.1 PXI Compliance Requirements for AXIe Components

RULE 5.2: AXIe System Modules, Chassis, and Peripheral modules shall comply with the corresponding requirements in PXI-6, Revision 1.3, subject to the explanations and caveats in Table 2, *Details of how PXI-6 bears on AXIe components.*

Note that PXI requires that the system controller provide a resource manager. Table 2, *Details of how PXI-6 bears on AXIe components* includes a column highlighting the PXI requirements for the resource manager. A PXI compliant resource manager will correctly enumerate an AXIe system and create a correct PXIESYS.INI file that includes the AXIe system.

In Table 2, *Details of how PXI-6 bears on AXIe components:*

Req Indicates that this requirement shall be satisfied by the module in the corresponding column

NA Indicates that this requirement is not applicable to the module in the corresponding column

<other> Other text in this field indicates the required behavior for the module in the corresponding column

Table 2 Details of how PXI-6 bears on AXIe components

PXI-6 Section	System Controller (Resource Manager)	System Module	Shelf/Chassis	Peripheral Module
2.1 Common File Requirements	NA	Req	Req	Req
2.1.1 Version Descriptor	NA	Req	NA	NA

PXI-6 Section	System Controller (Resource Manager)	System Module	Shelf/Chassis	Peripheral Module
2.2.1 System Description Definitions	Req	NA	NA	NA
2.2.2 Resource Manager Descriptor	Req	NA	NA	NA
2.2.3 System Descriptor	Req	NA	NA	NA
2.2.4 Chassis Descriptor	Req	NA	NA	NA
2.2.5 Trigger Bus Descriptor	Req	NA	NA	NA
2.2.6 Trigger Bridge Descriptor	Req	NA	NA	NA
2.2.7 Line Mapping Specification Descriptor	Reg	NA	NA	NA
2.2.8 Star System Timing Sets Descriptor	Req	NA	NA	NA
2.2.9 Star Trigger Descriptor	Req	NA	NA	NA
2.2.10.1 System Slot Descriptors	Req	NA	NA	NA
2.2.10.2 Peripheral Slot Descriptor	Req	NA	NA	NA
2.2.11.1 Single-Chassis PXI Express System	Req	NA	NA	NA
2.3 Chassis Description Files	NA	NA	Req	NA
2.3.2 Chassis Descriptor	NA	NA	Req	NA
2.3.3 Trigger Bus Descriptor	NA	NA	Req AXIe has a fixed single bus	NA

PXI-6 Section	System Controller (Resource Manager)	System Module	Shelf/Chassis	Peripheral Module
2.3.4 Trigger Bridge Descriptor	NA	NA	Req	NA
2.3.5 Line Mapping Specification Descriptor	NA	NA	Req	NA
2.3.6 Star System Timing Sets Descriptor	NA	NA	Required as described in section 5.3	NA
2.3.7 Star Trigger Descriptor	NA	NA	Required as described in section 5.3	NA
2.3.8 PXI-1 Bus Segment Descriptor	NA	NA	AXIe chassis shall NOT include a PXI-1 bus segment descriptor unless they have special provisions for PXI-1 devices	NA
2.3.9 Slot Descriptor	NA	NA	Req	NA
3.3.1 System Module Drivers	NA	Req	NA	NA
3.3.2 Chassis Drivers	NA	NA	Not required since AXIe devices are PCIe and this component is a PXI-1 requirement.	NA
3.3.3 Peripheral Module Drivers	NA	NA	NA	Req
3.3.4 Status Codes	NA	Req	Req	Req
3.4 Registration of Services	NA	Req	Req	Req
3.4.1 Services Tree	NA	Req	Req	Req
3.5 System Enumeration	NA	Req	NA	NA

PXI-6 Section	System Controller (Resource Manager)	System Module	Shelf/Chassis	Peripheral Module
3.5.1 Resource Manager Algorithm	Req	NA	NA	NA
3.5.2 Determining Chassis Numbers	Req	NA	NA	NA
3.5.3 Handling Driver Errors	NA	Req	NA	NA
4.2 PXI Software Compatibility (this section requires compliance with PXI-2, section 3)	NA	Req	Req	Req
4.3 32-bit Windows System Framework				
4.3.2 System Description File Location	NA	Req	NA	NA
4.3.3 System Configuration File Location	NA	Req	Req	NA
4.3.4 Chassis Description File Path Location	NA	Req	Req	NA
4.3.5 Driver Software Bindings	NA	Req	Req	Req
4.3.6 Services Tree Implementation	NA	Req	Req	Req
4.4 64-bit Windows System Framework				
4.4.2 System Description File Location	NA	Req	NA	NA
4.4.3 System Configuration File Location	NA	Req	Req	NA
4.4.4 Chassis Description File Path Location	NA	Req	Req	NA
4.4.5 Driver Software Bindings	NA	Req	Req	Req
4.4.6 Services Tree Implementation	NA	Req	Req	Req
4.5 32-bit Linux System Framework				

PXI-6 Section	System Controller (Resource Manager)	System Module	Shelf/Chassis	Peripheral Module
4.5.2 System Description File Location	NA	Req	NA	NA
4.5.3 System Configuration File Location	NA	Req	Req	NA
4.5.4 Chassis Description File Path Location	NA	Req	Req	NA
4.5.6 Services Tree Implementation	NA	Req	Req	Req
4.5.7 Security of PXI Files and Interfaces	NA	Req	Req	Req
4.6 64-Bit Linux System Framework				
4.6.2 System Description File Location	NA	Req	NA	NA
4.6.3 System Configuration File Location	NA	Req	Req	NA
4.6.4 Chassis Description File Location	NA	Req	Req	NA
4.6.5 Driver Software Bindings	NA	Req	Req	Req
4.6.6 Services Tree Implementation	NA	Req	Req	Req
4.6.7 Security of PXI Files and Interfaces	NA	Req	Req	Req
Appendix: 32-Bit Windows System Framework Files[2]	NA	Req	Req	Req

5.2 AXIe Slot Numbers, Physical Addresses, and Logical Addresses

All AXIe devices have multiple addresses including:

.

Physical Address This identifies the physical location of the module within the frame. AdvancedTCA® requires that the leftmost (or bottom) slot be 1 and increment from left to right (or bottom to top).

Logical Address A Slot within a Shelf defined by the Zone 1 Hardware Address. Every Slot has a unique Logical Slot number (maximum of 16 Logical Slots per Shelf). Logical Slot numbers are used to determine Channel mapping between Slots. PICMG® 3.0 defines a direct correlation between Channel numbers and Logical Slot numbers. For example, Channel 1 (Base and/or Fabric Channels) of every Slot establishes a direct connection to Logical Slot 1.

Hardware Address This address is set by using 7 lines directly connected to each module. This corresponds to the PXI geographical address. This same address is used as the IPMB address. The hardware address and physical address in AXIe differ numerically by 0x40 (64 decimal) (per AdvancedTCA® section 3.2.3.2).

For the purposes of this spec and relating AXIe to PXI, the slot number reported as the PXI slot number shall be the AXIe *physical address.*

RULE 5.3: When an AXIe peripheral module driver reports the module's slot number, it shall report the AXIe physical address.

5.2.1 Slot Number of the AXIe System Module

RULE 5.4: If the system module is not an embedded system module, it shall use its physical address as the PXI slot where the system module is located.

RULE 5.5: If the system module is an embedded system module, that is, the system module is embedded in the chassis, then the system module shall use a PXI slot of 0.

The address of the system slot appears in the system description file. It is available to the resource manager via the CompactPCI EPROM information.

Furthermore, since the AXIe System Slot provides timing functionality, the system slot should be reported as the timing module slot.

System software designed for PXI, where the system module is always at slot number 1, may have to be extended to properly support AXIe slot system module slot numbers which may be any slot, or zero in the case of an embedded system module.

5.3 Representing AXIe Triggers

RULE 5.6: AXIe description files shall use the system slot number as the PXI timing slot.

Table 3, *AXIe trigger representation in the PXI chassis description file*, describes how AXIe triggers shall be described to the PXI system software.

Table 3 AXIe trigger representation in the PXI chassis description file

Context	PXI definition and usage	AXIe

Trigger Bus Descriptor	Describes the electrically separate PXI trigger buses. Note that external control handles buffering between segments.	**RULE 5.7:** AXIe chassis TriggerBus descriptor shall always indicate a single non-segmented trigger bus to each slot.
		
		For example:
		`[TriggerBus1]`
		`SlotList = 1,2,3,4,5,6,`
		`...`
		
		Where the SlotList includes all slots in the chassis including the system module slot.

Star System Timing Sets Descriptor (that is the connections of the star triggers defined for PXI Express)	A single timing set descriptor is included for each timing slot in the chassis. Each indicates the mapping of the set of 3 PXI Express DTAR triggers between timing slot outputs and module DSTAR trigger inputs	**RULE 5.8:** The AXIe chassis shall not include a StarSystemTimingSet descriptor since the single AXIe star trigger is reported with a StarTriggerDescriptor.

Context	PXI definition and usage	AXIe

Star Trigger Descriptor (that is, the connection of the star trigger defined by PXI-1)	For each slot, a description of which timing slot trigger output is routed to which module slot. Note that the PXI-1 star trigger is a single bidirectional trigger much like AXIe.	**RULE 5.9:** The AXIe chassis StarTrigger descriptor always indicates a single star trigger to each slot, with the system module slot as the SystemTimingSlot.
		For example:
		`[StarTrigger0]`
		`SystemTimingSlot = 0`
		`PXI_STAR0=1`
		`PXI_STAR1=2`
		`...`
		The system timing slot point at the slot containing the system module or zero for embedded system modules.

5.4 System Module Software Requirements

The following sections describe software requirements regarding the system module.

5.4.1 Addressing the System Module and Chassis

RULE 5.10: The *address_info* value returned by the system module driver *PXISA_SystemModule_GetName* function shall return a string that a client can use to open a session to either a chassis or system module software programming interface (that is, an *instrument driver*) to both chassis and system module functions.

RULE 5.11: The *address_info* string is constructed by concatenating the VISA address of the chassis with the "+" character and the VISA address of the system module. In the case of an embedded system module, if the drivers for both the system module and the shelf may be opened using the same string, a single VISA address string may be used.

For instance, if an IVI driver is provided to control the chassis, the *address_info* string can be passed to the *Initialize* function to open a session to this particular chassis. Similarly, the same string can be passed to the *Initialize* function of the IVI driver for the system module to open a session with this system module.

For example, if the IP address of the chassis were 192.168.1.0 and the IP address of the system module were 192.168.1.1, an appropriate addressInfo response would be:

```
TCPIP::192.168.1.0::SOCKET+TCPIP::192.168.1.1::SOCKET
```

Note that the resource manager will place this string in the PXIESYS.INI file where it can be accessed by client software. Further, spaces are not allowed in the the *address_info* string.

5.4.2 Reporting Link Information

PXI systems may provide either 2 or 4 links to the system controller, each of varying width. These may be directly connected to a limited number of modules, or routed through PCI switches for more flexible connections. AXIe

system modules have a star topology, with 1 to 4 PCI Express connection to each slot. In order to interoperate with PXI systems and PXI system software, AXIe modules report artificial values.

The AXIe system module driver shall report 2 8-lane connections and associate at least one actual bus number and subordinate bus number with each reported link in order to respond to the various queries issued with the *PXISA_SystemModule_GetInformation* call.

RULE 5.12: The system module driver *PXISA_SystemModule_GetInformation* call shall indicate 2, 8-lane links. The bus number range reported for these two links shall both be non-empty. All of the bus numbers of the links to the instrument modules shall be associated with one and only one of the two links.

5.4.3 Reporting the Compact PCI EEPROM

RULE 5.13: System Modules that only implement AXIe-1 versions before Rev 3 shall provide a system module driver. The *PXISA_SystemModule_GetChassisEeprom* shall return a data structure constructed to mimic a Compact PCI EEPROM. Specifically, the following fields shall reflect the corresponding AXIe chassis information:

- Vendor ID (string)
- Model (string)
- Serial Number (string)
- Slot Count (byte)
- Peripheral Slot Count (physical number of peripheral slots)
- System Slot number, that is, the physical address of the system slot (byte)
- System Slot type (AXIe shall always report a 2-Link slot type)
- System Slot Link 1 width (AXIe shall always report an 8-lane link)
- System Slot Link 2 width (AXIe shall always report an 8 to indicate an 8-lane secondary link)
- For each slot, individual slot link connectivity
  - Peripheral slot number, that is, the physical location
  - Link number 1 origin (that is, the link number to the system slot)
  - Link number 2 origin as “no connection”
  - Slot type
  - Peripheral slot link 1 width
  - Peripheral slot link 2 width shall be reported as “Not routed”

RULE 5.14: <Provisional addition to AXIe-2.0 Revision 2.0> System Modules that only implement AXIe-1 versions before Rev 3 shall provide a system module driver. The *PXISA_SystemModule_GetChassisEeprom* shall return a data structure constructed to mimic a Compact PCI EEPROM. Specifically, the following fields shall reflect the corresponding AXIe chassis information:

System Modules that support AXIe-1 Rev 3 and newer shall report:

- Vendor ID (string)
- Model (string)
- Serial Number (string)
- Slot Count (byte)
- Peripheral Slot Count (physical number of peripheral slots)
- System Slot number, that is, the physical address of the system slot (byte)
- System Slot type (AXIe shall always report a 2-Link slot type)
- System Slot Link 1 width (AXIe shall always report an 8-lane link)
- System Slot Link 2 width (AXIe shall always report an 8 to indicate an 8-lane secondary link)

- For each slot, individual slot link connectivity
  - Peripheral slot number, that is, the physical location
  - Link number 1 origin (that is, the link number to the system slot)
  - Link number 2 origin (that is, the link number to the system slot)
  - Slot type
  - Peripheral slot link 1 width
  - Peripheral slot link 2 width

Note that information regarding AXIe Links 3 and 4 are not reported via this mechanism.

Details of the format for this information are in the PICMG CompactPCI specification.

5.4.4 Behavior of the SMBusOperation Function

RULE 5.15: The system module driver *PXISA_SystemModule_SMBusOperation* shall not return an error although no equivalent SMBus operation is provided to the chassis.

5.5 Chassis Software Requirements

In the PXI Architecture, a chassis driver is used to enumerate the PXI-1 buses in each chassis. Since an AXIe chassis manifests as a pure PXI Express chassis, AXIe chassis are not required to provide a PXI chassis driver.

RULE 5.16: The chassis description file for an AXIe chassis shall indicate the slot connected by the local bus connection to the left and right.

Note that for AXIe, this does not indicate the width of the connection.

The following example shows a valid descriptor of the adjacent slots to an AXIe slot:

```
[SLOT4]
LOCALBUSLEFT = "SLOT3"
LOCALBUSRIGHT = "SLOT5"
```

5.5.1 An Example Chassis Description File

The following example shows a valid AXIe Chassis Description File:

```
# Example Chassis Description file.

[Chassis]
Model = "M9505a"
Vendor = "Agilent Technologies"
Slots = "1,2,3,4,5"
TriggerBusList = "1"
StarSystemTimingSetList = "1"

[TriggerBus1]
SlotList = "0,1,2,3,4,5"

[StarTrigger1]
SystemTimingSlot = 0   # ESM is part of chassis, referred to as "slot 0"
PXI_STAR0 = 1
PXI_STAR 1 = 2
PXI_STAR 2 = 3
PXI_STAR 3 = 4
PXI_STAR 4 = 5
```



```
[Slot1]
LocalBusLeft = "None"
LocalBusRight = "Slot2"

[Slot2]
LocalBusLeft = "Slot1"
LocalBusRight = "Slot3"

[Slot3]
LocalBusLeft = "Slot2"
LocalBusRight = "Slot4"

[Slot4]
LocalBusLeft = "Slot3"
LocalBusRight = "Slot5"

[Slot5]
LocalBusLeft = "Slot4"
LocalBusRight = "None"
```

5.6 Peripheral Module Requirements

AXIe modules may include peripheral module description files. In which case, the resource manager will treat them exactly as PXI peripheral module description files.

RULE 5.17: The peripheral module driver *PXISA_PeripheralModule_GetInformation* call shall return the actual maximum link width supported by the module, up to the AXIe imposed maximum.

RULE 5.18: The peripheral module driver *PXISA_PeripheralModule_GetInformation* call shall return the negotiated link width and PCI bus number as required by the PXI-6 specification.

RULE 5.19: The peripheral module driver *PXISA_PeripheralModule_GetInformation* call shall return the AXIe logical address when the client queries for slot number.

6 Additional AXIe Software Requirements

This section has general requirements not directly associated with other sections.

6.1 Instrument Driver Requirement

RULE 6.1: All AXIe devices that support the PXI 32 or 64-bit Windows framework shall provide an IVI Specific Driver. The details of this requirement are called out in Section 5 of IVI-3.1.

RULE 6.2: All AXIe devices that support the PXI 32 or 64-bit Linux framework shall provide a software driver consistent with the PXI requirements.

Devices may optionally provide additional drivers. This is especially appropriate for environments other than Microsoft Windows.

6.2 Additional AXIe Software Requirements

RULE 6.3: < Provisional addition to AXIe-2.0 Revision 1.1 > This section defines a set of application program interfaces (APIs) that shall be provided by AXIe system modules to access various FRU information. These API's are provided as additional functions of the PXI-6 defined system module driver.

These functions provide access to the FRU component fields defined by IPMI 1.0 of the AXIe modules.

RULE 6.4: Calls to these functions are only guaranteed to work after calling the PXI system module driver *PXISA_SystemModule_GetCount* function.

RULE 6.5: Any cached information associated with these functions shall be refreshed when *PXISA_SystemModule_GetCount* is called.

RULE 6.6: If any of the AXIe PXI System Module Driver APIs defined in this section are implemented, all shall be implemented.

RULE 6.7: Calling conventions, data type definitions, return value definitions, bitness, and installation shall track the PXI-6 specifications regarding the implementation of PXI defined functions of the chassis driver.

6.2.1 GetVersion

Get the revision number number of the AXIe standard that this API complies with. The info string permits the system module driver to return an arbitrary vendor defined string.

```
tPXISA_Status AXIe_GetVersion(
   __out    tPXISA_PInteger   version,
   __out    tPXISA_PString    info
);
```

Parameters

version	An integer specifyign the version of the AXIe spec this API complies with. For version 1.0 of the specification, 0x00010000 shall be returned.
info	An arbitrary string that may be used by the system module driver to communicate other vendor specific information.

Return Value

This operation shall always complete successfully, the return value is *kPXISA_Success*.

6.2.2 GetLUNCount

Get the number of LUNs (Logical UNits) associated with the device in the specified slot. GetComponentCount is permitted to cache information associated with the chassis and slot to optimize GetComponentCount however it is required to clear any cached information when it is called.

```
tPXISA_Status AXIe_GetLUNCount(
   __in     tPXISA_Integer    chassisNumber,
   __in     tPXISA_Integer    slotNumber,
   __out    tPXISA_Pinteger   LUNcount
);
```

Parameters

chassisNumber	The chassis number containing the instrument module, as returned using the PXI-6 *PXISA_SystemModule_GetCount* function.
slotNumber	The slot number of the instrument module to query for LUN Component Count. This is the AXIe physical slot number.
LUNcount	The number of LUNs on the instrument module in the designated slot. A count of 0 shall indicate an empty slot.

Return Value

If the operation completes successfully, the return value is *kPXISA_Success*. If the driver has not been properly initialized (for instance because the client failed to call *PXISA_SystemModule_GetCount*), *kPXISA_Error* is returned.

If the ShMC is unable to communicate with the instrument module identified (for instance, if the slot is empty), *kPXISA_Warning* is returned, and the count is set to zero.

6.2.3 GetComponentCount

Gets the number of components provided by a given LUN on the module in the specified slot of the specified chassis. GetComponentCount is permitted to cache information associated with the chassis, slot, and LUN to optimize GetFieldCount; however it is required to clear any cached information when it is called.

```
tPXISA_Status AXIe_GetComponentCount(
   __in     tPXISA_Integer    chassisNumber,
   __in     tPXISA_Integer    slotNumber,
   __in     tPXISA_Integer    LUNIndex,
   __out    tPXISA_String     componentCount
);
```

Parameters

chassisNumber	The chassis number containing the instrument module, as returned using the PXI-6 *PXISA_SystemModule_GetCount* function.
slotNumber	The physical slot number of the instrument module to query.
LUNIndex	The zero-based index of the LUN on the instrument module.
componentCount	The number of components associated with this FRU.

Return Value

If the operation completes successfully, the return value is *kPXISA_Success*. If the driver has not been properly initialized (for instance because the client failed to call *PXISA_SystemModule_GetCount*), an error is returned.

If the ShMC is unable to communicate with the instrument module identified (for instance, if the slot is empty), an error is returned.

If the instrument module does not return a value for the queried field, an error is returned.

6.2.4 GetFieldCount

Gets the number of fields and component name of a given component of the LUN on the module in the specified slot of the specified chassis. GetFieldCount is permitted to cache information associated with the chassis, slot, LUN, and component to optimize GetFieldInfo; however it is required to clear any cached information when it is called.

```
tPXISA_Status AXIe_GetFieldCount(
    __in     tPXISA_Integer    chassisNumber,
    __in     tPXISA_Integer    slotNumber,
    __in     tPXISA_Integer    LUNIndex,
    __in     tPXISA_Integer    componentIndex,
    __out    tPXISA_String     componentName,
    __out    tPXISA_Integer    fieldCount
);
```

Parameters

chassisNumber	The chassis number containing the instrument module, as returned using the PXI-6 *PXISA_SystemModule_GetCount* function.
slotNumber	The physical slot number of the instrument module to query.
LUNIndex	The zero-based index of the LUN on the instrument module.
componentIndex	The zero-based index of the component of the LUN.
componentName	The name of the component identified by chassisNumber, slotNumber, LUNIndex, and componentIndex
fieldCount	The number of fields associated with this component

Return Value

If the operation completes successfully, the return value is *kPXISA_Success*. If the driver has not been properly initialized (for instance because the client failed to call *PXISA_SystemModule_GetCount*), an error is returned.

If the ShMC is unable to communicate with the instrument module identified (for instance, if the slot is empty), an error is returned.

If the instrument module does not return a value for the queried field, an error is returned.

6.2.5 GetFieldInfo

Get the field information for the component designated by the chassis number, slot number, LUN, FRU and component index. This may only be called after calling GetFieldCount for the chassis and slot being queried. This allows information to be gathered by GetFieldCount and cached for calls to this function.

```
tPXISA_Status AXIe_GetFieldInfo(
    __in     tPXISA_Integer    chassisNumber,
    __in     tPXISA_Integer    slotNumber,
    __in     tPXISA_Integer    LUNIndex,
    __in     tPXISA_Integer    componentIndex,
    __in     tPXISA_Integer    fieldIndex,
```



```
    __out    tPXISA_String     fieldName,
    __out    tPXISA_String     fieldValue
);
```

Parameters

chassisNumber	The chassis number containing the instrument module, as returned using the PXI-6 *PXISA_SystemModule_GetCount* function.
slotNumber	The physical slot number of the system to query
LUNIndex	The zero-based index of the LUN on the instrument module.
componentIndex	The zero-based index of the component on the FRU.
fieldIndex	The zero-based index of the field of the FRU on the instrument module.
fieldName	The name of the queried FRU field
fieldValue	The value of the queried FRU field

Return Value

If the operation completes successfully, the return value is *kPXISA_Success*. If the driver has not been properly initialized (for instance because the client failed to call *PXISA_SystemModule_GetCount*), an error is returned.

If the ShMC is unable to communicate with the instrument module identified (for instance, if the slot is empty), an error is returned.

If the instrument module does not return a value for the queried field, an error is returned.

6.2.6 GetLinkOrigins

Gets the origin of the four links provided to a given slot. That is, the system module link that provides the connection for each peripheral module link.

The response parameters indicate which of the system module channels are associated with each link.

The origin shall be reported as -1 for any link that is not active.

```
tPXISA_Status AXIe_GetLinkOrigins(
    __in     tPXISA_Integer    chassisNumber,
    __in     tPXISA_Integer    slotNumber,
    __out    tPXISA_Integer *  originOfLink1,
    __out    tPXISA_Integer *  originOfLink2,
    __out    tPXISA_Integer*   originOfLink3,
    __out    tPXISA_Integer*   originOfLink4
);
```

Parameters

chassisNumber	The chassis number containing the instrument module, as returned using the PXI-6 *PXISA_SystemModule_GetCount* function.
slotNumber	The physical slot number of the instrument module to query.
originOfLink1	The system module channel associated with Link 1.
originOfLink2	The system module channel associated with Link 2.
originOfLink3	The system module channel associated with Link 3.
originOfLink4	The system module channel associated with Link 4.

Return Value

If the operation completes successfully, the return value is *kPXISA_Success*. If the driver has not been properly initialized (for instance because the client failed to call *PXISA_SystemModule_GetCount*), an error is returned.

6.2.7 GetLinkWidths

Gets the of number of lanes in each of the four links available to a given slot.

Note that system-module-specific mechanisms are used to configure the fabric that the system module provides. This API queries the currently active configuration.

The response parameters that return the number of lanes in the link use the designations specified in PICMG EXP.0 specification. That is:

0h—Not routed
1h—x1 2.5 GT/s PCI Express Signaling
2h—x4 2.5 GT/s PCI Express Signaling
3h—x8 2.5 GT/s PCI Express Signaling
4h—x16 2.5 GT/s PCI Express Signaling
5h—x1 5.0 GT/s PCI Express Signaling
6h—x4 5.0 GT/s PCI Express Signaling
7h—x8 5.0 GT/s PCI Express Signaling
8h—x16 5.0 GT/s PCI Express Signaling
9h—x1 8.0 GT/sc PCI Express Signaling
Ah—x4 8.0 GT/s PCI Express Signaling
Bh—x8 8.0 GT/s PCI Express Signaling
Ch—x16 8.0 GT/s PCI Express Signaling
Dh-Maximum integer — Reserved
Minimum integer - -1 — Reserved

```
tPXISA_Status AXIe_GetLinkWidths(
   __in     tPXISA_Integer    chassisNumber,
   __in     tPXISA_Integer    slotNumber,
   __out    tPXISA_Integer *  lanesInLink1,
   __out    tPXISA_Integer *  lanesInLink2,
   __out    tPXISA_Integer *  lanesInLink3,
   __out    tPXISA_Integer *  lanesInLink4
);
```

Parameters

chassisNumber	The chassis number containing the instrument module, as returned using the PXI-6 *PXISA_SystemModule_GetCount* function.
slotNumber	The physical slot number of the instrument module to query.
lanesInLink1	The number of PCI Express lanes configured for Link 1.
lanesInLink2	The number of PCI Express lanes configured for Link 2.
lanesInLink3	The number of PCI Express lanes configured for Link 3.
lanesInLink4	The number of PCI Express lanes configured for Link 4.

Return Value

If the operation completes successfully, the return value is *kPXISA_Success*. If the driver has not been properly initialized (for instance because the client failed to call *PXISA_SystemModule_GetCount*), an error is returned.

6.2.8 Data types

All data types used in this API are defined by PXI-6.

Note that all strings are allocated by the client.

A. Summary of the PXI Software Architecture

The AXIe software architecture relies heavily on the PXI software requirements. This appendix details the PXI Software Architecture.

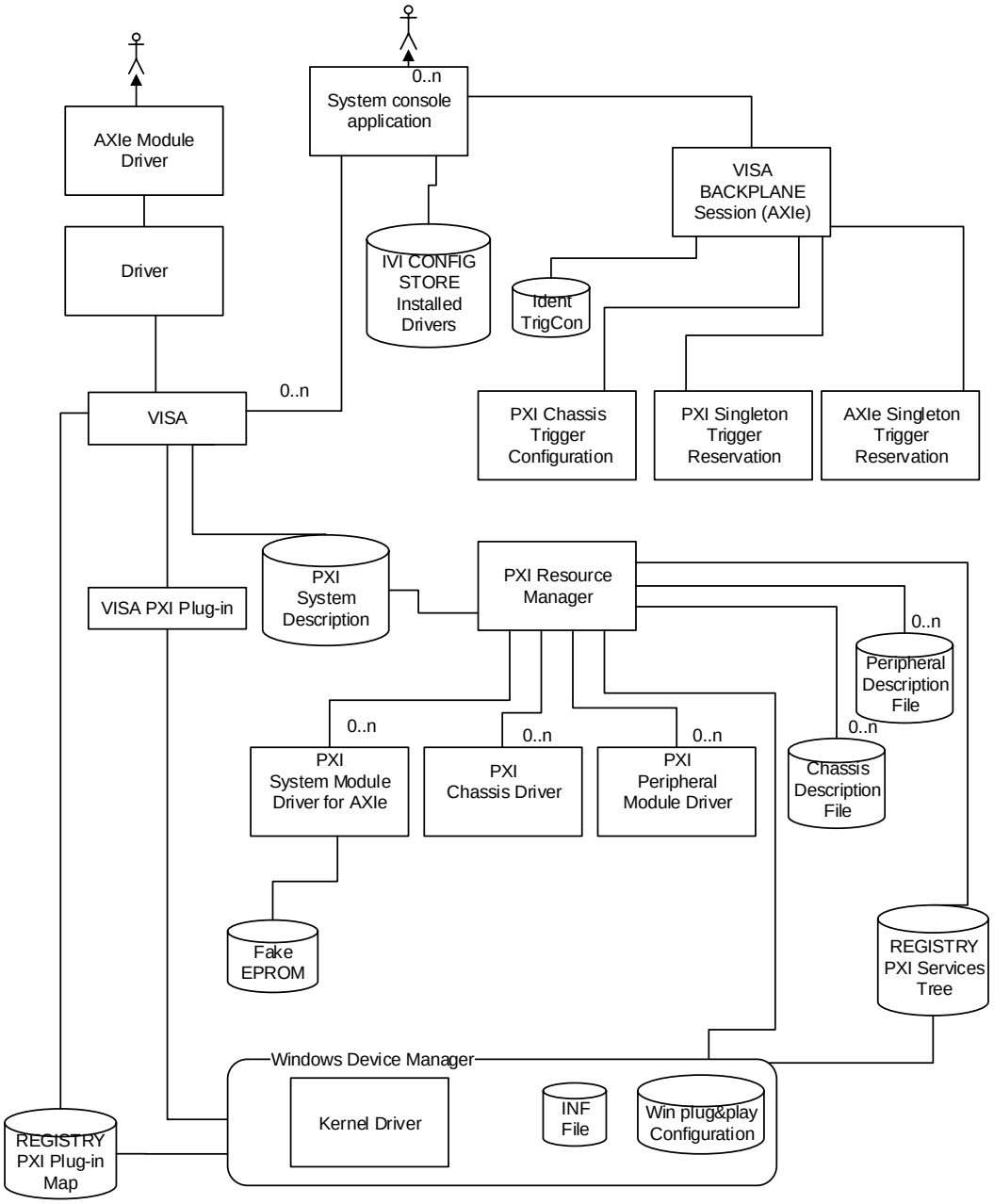


Figure 4 Basic PXI Software Architecture

The purpose of the PXI resource manager is to create a manifest of all of the system modules, chassis, and peripheral modules on the system. This manifest contains the type of modules discovered and the addressing information necessary to communicate with them. Most importantly, the manifest provides the physical location of the modules, this is necessary for client applications to distinguish between identical hardware modules. The manifest is written to the Windows root directory in two pieces, one for the parallel PXI-1 devices, called

PXISYS.INI and one for the PXI Express devices, including AXIe, called PXIESYS.INI. Using this information an IO library, such as VISA, is able to conduct IO to module by specifying its physical location.

The PXI software specifications define the software provided with PXI hardware that is necessary for the resource manager to work with an arbitrary vendor's hardware. The necessary software is made up of drivers and static description files provided with various hardware components. These components discover and describe the corresponding hardware components. The resource manager uses these drivers and description files whenever the system is enumerated.

A.1 Overview of PXI System Enumeration

PXI system enumeration is the mechanism where a manifest of the currently connected PXI modules is constructed. The enumeration process creates a file with an entry for every PXI module connected to the system including its physical location (chassis designation and slot number), and the information needed to conduct IO with it. Per the PXI specifications, the PXI system enumeration is done by the *resource manager*. PXI-6 specifies the details of how this is done.

The PXI resource manager and the surrounding architecture are illustrated in Figure 4, *Basic PXI Software Architecture.*

To enumerate PXI Express, including AXIe resources, the resource manager uses system module drivers and peripheral module drivers to gather the dynamic information (such as the bus number assigned to a device by the operating system) and the chassis description file for static information.

PXI requires that each system module and peripheral module installed in the system have corresponding software installed on the controlller. As part of the software install, the peripheral and system modules create a list in the registry of the hardware that may be installed on the system. The list includes the vendor name and model name of the hardware and a PXISA specified driver.

During system enumeration the resource manager calls the driver for each piece of hardware that is potentially installed on the system. Each driver reports to the resource manager how many instances of the hardware is present along with addressing information and the physical location of instrument modules.

Using this information, the resource manager builds a manifest that includes:

- All the PXI Express modules, chassis, and system modules installed on the system
- The chassis that each module is installed
- The geographical location of modules within the chasssis (that is, the slot number)
- The system module that accesses them
- Addressing information for each module
- Static information about the chassis such as the star trigger configuration

A.2 System enumeration by the Resource Manager

The algorithm followed by the resource manager is in the PXI-6 specification (sect 3.5.1). Software drivers are defined for the system modules and peripheral modules. These must be provided by all PXI and AXIe hardware providers. In addition, PXI specifies a chassis driver, however, for PXI Express devices, static description files provide all of the necessary information. Therefore chassis that are exclusively PXI Express, such as AXIe, may omit the chassis driver.

The resource manager calls each system module driver installed on the system.[3] This provides a list of chassis, and the range of PCI bus numbers contained in each chassis. The resource manager then calls each of the peripheral

[3] Although not explicitly tagged as rules in the referenced PXI standard, the code files listed in the appendixes define some values that are critical for interoperability.

module drivers installed on the system. This provides a list of all of the peripherals installed along with their PCI bus number and slot number (that is, the geographical address).

The resource manager combines this information to create the system manifest containing the list of chassis, the modules installed on each, the slot number of each, and the information necessary to address each device.

A.3 The PXI Services Tree

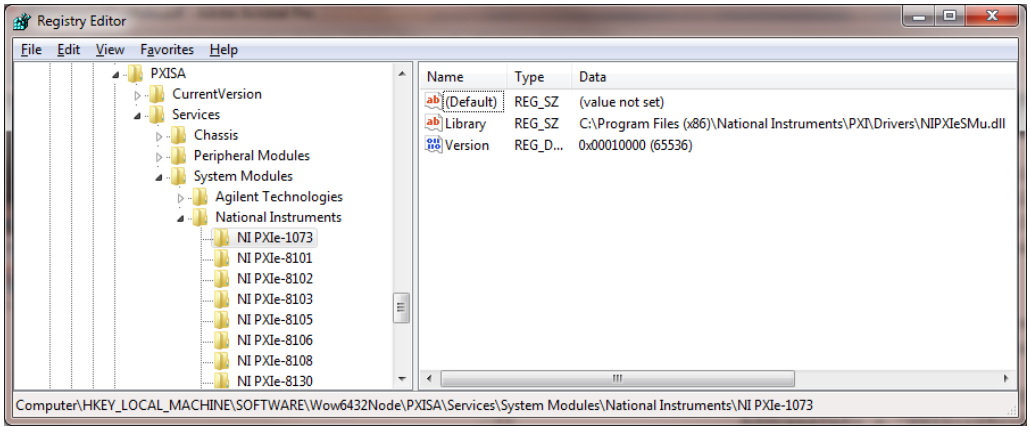


Figure 5 Typical PXI services tree showing the NI PXIe-1073 system module

The PXI services tree is a registry structure that contains a list of all installed hardware and the system module drivers, chassis drivers, and peripheral module drivers that support it. There are keys for each type of driver, and under each key is a list of vendor names, and under each vendor name is a list of model names which contains the references to the PXI defined driver DLL's. Figure 5, *Typical PXI services tree showing the* , illustrates a typical system.

Note that each driver is called with the vendor name key and the model name key under which the driver is located in the registry. For instance, the illustrated system driver will be passed "National Instruments" as the vendor name, and "NI PXIe-1073" as the model. By providing this information to the driver, the same driver can be used to support equipment from different vendors and of different models.

A.4 The PXI System Module driver

As discussed above, the PXI System Module driver is used by the resource manager to build a list of the system modules on the system along with the modules plugged into it. As such, most of the work of enumerating the system is done by this driver.

Table 4, *the PXI System Module API and its use*, describes the API and its usage for AXIe.

Table 4 the PXI System Module API and its use

`Status PXISA_SystemModule_GetCount(String vendor, String model, Integer * count)` **vendor**: Vendor name to match. **model**: Model name to match. **count**: Returns Number of System Modules found by the driver.
The resource manager traverses the PXI services tree in the registry and calls each installed system module driver as the first step to building the list of resources. The system module drivers are used by first calling

GetCount to determine the number of installed system modules, then the *GetName* and *GetInformation* API's are used to get additional information about each discovered system modules.

The resource manager passes *GetCount* the vendor name key and model name key that are currently being enumerated. By including these values, the same peripheral driver can be used for different system modules.

Typically a PXI implementation of *GetCount* will call the device manager to locate the system modules that the operating system discovered and configured. For instance, using the Windows Driver Development kit device installation API's such as *SetupDiEnumDeviceInfo()*. An AXIe implementation will need to use an approach specific to the system module to detect and enumerate its system modules.

Note that as the PXI resource manager is enumerating system modules supported by this system module driver. It begins by calling this function, *GetCount*, and then calls the *GetName* and *GetInformation* functions for all values of the index starting at one to the count. Therefore, a logical implementation is to capture and cache all of the necessary information when the *GetCount* function is called and return the name and other information from that cache on subsequent calls.

PXI-6 requires that any cached information is cleared when *GetCount* is called.

```
Status PXISA_SystemModule_GetName(String vendor, String model, Integer index,
String * name, String * addressInfo)
```

vendor: Vendor name to match.
model: Model name to match.
index: Index of a System Module. This index is 1-based.
name: Unique name of a System Module.
addressInfo: Additional addressing information for the module

As with *GetCount*, *GetName* is passed the vendor and model keys that were associated with this driver in the registry. *GetName* is also passed an index from 1 to the count returned by *GetCount* to get the name and address for each discovered system module.

GetName provides identifying information for this system module. Note that the name could reasonably be a constant name followed by an integer designating the instance of the system module.

For PXI, *addressInfo* is a string that contains any necessary address information required by the *SMBusOperation* function in this driver. That is, this *addressInfo* string will be passed to any subsequent calls to *SMBusOperation.*

Although AXIe devices must comply with this requirement, AXIe *RULE 5.10:*further requires that this same address string may be used to initialize the instrument driver to communicate with this instance of a system module, or with the connected chassis.

For AXIe the VISA address of the system module is a VISA LAN address of the form:

```
TCPIP0::host address::port::SOCKET
```

For instance:

```
TCPIP0::192.168.1.13::5025::SOCKET
TCPIP0::MyAXIeChassis::5025::SOCKET
```

Note that in both examples, port 5025 is shown. In the first example, the numeric IP address and port are explicit, and in the second example a hostname is used for the IP address.The PXI Resource Manager will record this address string in the system description file for subsequent use by the client.

For an AXIe embedded system module may just be this system module address. For system modules that are not embedded, an additional address for the shelf is included with a '+' separator between the two. Thus the response is <chassis address>+<system module address>. For instance:


```
6.2.8.1.1.1.1.1 TCPIP0::192.168.1.13::5024::SOCKET+TCPIP0::192.168.1.14::
5024::SOCKET
```

```
Status PXISA_SystemModule_GetInformation(String name, String addressInfo,
Integer field, Integer * value);
```
name: Unique name of a System Module.
addressInfo: Additional addressing information for the System Module.
field: Selector for which information field is requested.
value: Value of the information field.

As with *GetCount*, *GetInformation* is passed the vendor and model keys that were associated with this driver in the registry. *GetInformation* interrogates the system module driver for the link configuration. This includes:

- The number of lanes in each link
- The bus number and subordinate bus numbers of each PCI link (note that the PXI system module connectors support a maximum of four links).

The information being queried is specified with the *field* parameter and is specified in PXI-6, sect 3.3.1. The response is the integer *value*.

For PXI, the resource manager uses this API to determine how many PCIe links exist to this system module, and the bus address range supported by each link (that is, the link bus number and the subordinate bus number). When the resource manager subsequently invokes the PXI Peripheral Drivers, it will determine which link (and therefore which chassis) each peripheral is located in.

Since the AXIe architecture provides a star PCIe connection from the system module to the instrument slots, each slot has its own link. In order for AXIe system modules to respond to this PXI query, AXIe system module drivers:

- Contrive values by reporting a number of links that is valid for PXI. Specifically, two are reported.
- Associating each actual AXIe device bus number with the fictitous PXI links and report appropriate subordinate bus numberes (for instance, the first link could report the first device and second link could report a bus number and subordinate bus number that spans all the remaining devices).

It is important to realize that those PCIe bus numbers, although valid for device communication, are not necessarily associated with a PCIe link through a single physical PCIe switch. For instance, in reality, the link implemented on the system module may use a single PCIe switch to reach all of the AXIe slots.

AXIe *RULE 5.12:* describes how AXIe devices respond to this query.

```
Status PXISA_SystemModule_GetChassisEeprom(String name, String addressInfo,
Buffer * chassisEeprom);
```
name: Unique name of a System Module.
addressInfo: Additional addressing information for the System Module.
chassisEeprom: Contents of the Chassis EEPROM. This buffer must be 256 bytes

This returns the Compact PCI specified EEPROM contents.

The format of the returned data is defined by the Compact PCI standard. It is used by the PXI resource manager to determine the topology of the parallel PCI devices. This resource manager uses this to determine the slot number of PXI-1 devices.

The EEPROM is also used to identify the chassis controlled by this system module driver. Specifically, the EEPROM indicates the serial number, vendor and model of the chassis. The resource manager uses these values to locate the chassis INI file in the registry and get additional chassis information such as the configuration of triggers within the chassis.

AXIe *RULE 5.12:* describes how AXIe devices respond to this query.

```
Status PXISA_SystemModule_SMBusOperation(String name, String addressInfo,
Integer protocol, Integer address, Integer command, Integer packetErrorCode,
Integer writeBufferCount, Buffer writeBuffer, Integer * readBufferCount, Buffer
* readBuffer);
```

name: Unique name of a System Module.
addressInfo: Additional addressing information for the System Module.
protocol: Protocol used for the SMBus operation
address: Address Byte sent to the device.
command: Command byte to send to the device.
packetErrorCode: Flag to specify whether to include the Packet Error Code. The value of this field should be zero (0) when the PEC should not be included, and one (1) when the PEC should be included.
writeBufferCount: Number of bytes to be written for Write Block commands.
writeBuffer: Data content of the operation.
readBufferCount: Number of bytes received for Read Block commands.
readBuffer: Data content received by the operation. This buffer must be 32 bytes.

For PXI Chassis this function is used to initiate arbitrary SMBus operations.

In general this is not used by the resource manager, however the function is available for applications that need to access the SMBus.

Since the SMBus is not supported with AXIe, this should return a warning and take no action for AXIe. It should not return an error because errors may cause the PXI resource manager to abort.

A.5 The PXI Express Peripheral Module driver.

All PXI Express peripheral modules (that is, instrument modules) are required to provide a PXI Express peripheral driver. The PXI Express peripheral module driver is responsible for:

- Enumerating all of the peripheral modules of a given model from a given vendor
- Providing the PCI bus number of each enumerated device
- Providing the slot number of each enumerated device
- Providing the maximum link width and negotiated link width of each enumerated device

The resource manager will correlate this information with the information returned by the system drivers to determine which chassis and system controller the module resides in.

The implementation of the PXI Express Peripheral Module driver uses the OS device manager to enumerate and provide PCI bus numbers of all devices that have been associated with a given kernel driver and PCI device ID. Given this bus number, the resource manager is able to look at the bus number and subordinate bus number of each link in each chassis and determine which chassis this peripheral module is in.


```
Status PXISA_PeripheralModule_GetCount(String vendor, String model, Integer *
count)
```

vendor: Vendor name to match.
model: Model name to match.
pmCount: Number of Peripheral Modules found by the driver, matching the criteria of the other parameters

This is the first call issued by the resource manager when using a peripheral module driver. As with the system module driver, *GetCount* is used to determine the number of modules in the system from this vendor, and of this model.

As with the PXI System Module driver, this will typically utilize the Windows Driver Development kit device installation API's.

The implementation will typically specify the GUID associated with a given device driver and build a list of peripherals of the given vendor and model.

Note that as the PXI resource manager is enumerating the peripheral modules supported by this peripheral module driver, it begins by calling this function, and then calls the *GetName* and *GetInformation* functions for all values of the index from one to the count. Therefore, a logical implementation is to capture and cache all of the necessary information when the *GetCount* function is called and return the name, and other information from that cache on subsequent calls. The PXI specs require that the cache is cleared when this function is called.

```
Status PXISA_PeripheralModule_GetName(String vendor, String model, Integer
index, String * name, String * addressInfo)
```

vendor: Vendor name to match.
model: Model name to match.
index: Index of a Peripheral Module. This index is 1-based.
name: Unique name of a Peripheral Module.
addressInfo: Additional addressing information for the module.

The resource manager uses this function to read back the module name and module addressInfo from each module reported by the *GetCount* function.

The returned name identifies this device (for instance "myDevice2"). The returned addressInfo is passed to the *GetInformation* function.

This is NOT necessarily the same string the resource manager writes to these values in the system description file where this module is identified.

```
Status PXISA_PeripheralModule_GetInformation(String name, String addressInfo,
Integer field, Integer * value);
```

name: Unique name of a Peripheral Module.
addressInfo: Additional addressing information for the Peripheral Module.
field: Selector for which information field is requested.
value: Value of the information field.

The resource manager uses this function to determine additional PCI information and geographic address of this module. Specifically, the slot number, maximum link width, PCI Bus Number, and negotiated link width are read back from the driver.

The resource manager passes this function the name and address string returned by *GetName*.

The field parameter designates the specific value that the resource manager is querying.

Note that AXIe devices return the logical address as the slot number.

A.6 Implementation of the Peripheral Module Driver

Much of the information required for AXIe enumeration is available at compile time. For instance, for a given endpoints the, PCI vendor-IDs, product IDs, product slot count, and end-point offset within the module is known. Consider:

- The module provider knows the module's model name for each endpoint (for example, KtM1234B)
- The module provider knows the occupied slot count and slot number offset (for example, if it is the middle slot of a 3-slot instrument). Note that this does prevent the use of identical software when installing a module stand-alone and embedding it into another device. However, that requirement is a direct fall-out of the specification, because otherwise, the module would incorrectly report the occupied slot count.
- The module provider knows the the endpoint resides in a multi-endpoint module, and if so, if it is the primary endpoint or a secondary endpoint (or beyond).

A single peripheral module drivers can support multiple modules. To do this, the peripheral module driver:

- Calls the operating system (in this case of Windows, this is the device manager) for information about the PCIe configuration
- Reads data provided when the module is installed that allows the peripheral module driver to associate the discovered endpoints with physical modules.

To support multi-link/multi-endpoint/multi-slot modules, the peripheral module driver reads additional configuration information provided when the driver software is installed.

A.6.1 *Peripheral Module Driver Behavior for Multi-Slot Modules*

For multi-slot modules, the peripheral module driver reports back the appropriate slot number offset and occupied slot count based its communication with the operating system PCI system and knowledge of the module from the module installation data.

A.6.2 *Peripheral Module Driver Algorithm for Multi-Endpoint Modules*

In order for the peripheral module driver to report back a full address_info string for a multi-endpoint module, it must:

- Enumerate all the endpoints for this module type, and
- Properly correlate them into individual modules (reporting the multiple addresses in a comma separated string)

To do this, the peripheral module driver can:

- Identify all installed drivers based on the driver GUID (note that an instance of a driver will be available for each endpoint.
- Reads back the slot information from each endpoint (by using the kernel driver configured for the device)

Next, the peripheral module driver needs to determine which endpoints are in the same module. For example, there could be several instances of a given multi-endpoint module. They could be plugged into different chassis (perhaps in the same slot) or the same chassis. If they were all plugged into a single chassis, the slot number could be used to associate them together. However, that is not adequate for multi-chassis systems. Therefore, the peripheral module driver needs to read PCIe topology information back from the operating system to determine which end-points are in the same module.

A.6.3 *Peripheral Module Driver Algorithm for Multi-Link Modules*

The peripheral module driver does not need any special handling for multi-link modules. Each slot enumerates conventionally, reporting the registry/INF data for the slot number offset and occupied slot count. It is the resource manager's responsibility to recognize these endpoints as being part of a multi-link module based on the geographical location.

A.7 The Chassis Description File

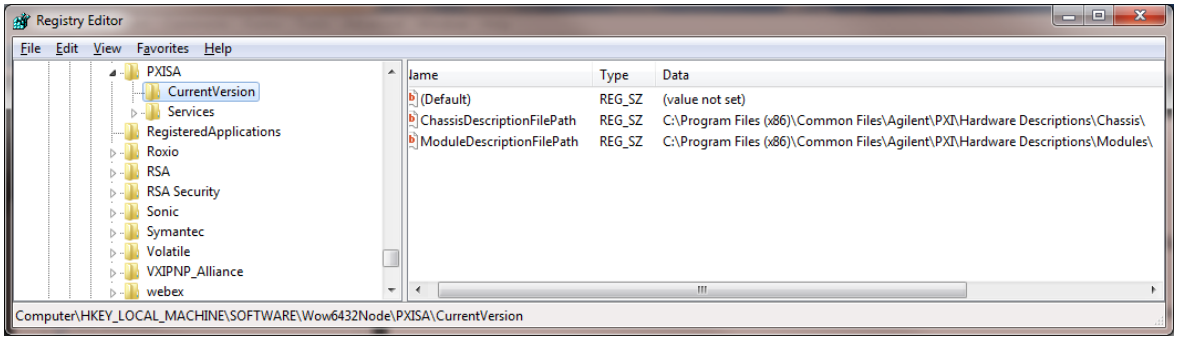


Figure 6 The registry entries made by the resource manager that specify where the chassis and module description files should be installed

The chassis description file is specified by PXI-6, section 2.3. It contains static information about the chassis that is not available from the Compact PCI EEPROM (which is available to the resource manager via the system module driver).

When the resource manager is installed, it specifies where the chassis description files should be installed. It does this by writing the directory value *ChassisDescriptionFilePath* to the registry key:

Key: HKEY_LOCAL_MACHINE\SOFTWARE\PXISA\CurrentVersion

Value: ChassisDescriptionFilePath

Figure 6 shows an example. Note that the registry key is located in the 32-bit registry hive on both 32 and 64-bit operating systems.

PXI-6 specifies that a chassis description file shall be provided by the chassis manufacturer and installed to the appropriate directory. PXI-6 specifies that the name of the installed description file shall be chassis_*vendorDefinedText*.ini. Since these files are all installed to the same directory, it is recommended to include the vendor name in this string to guarantee there are no conflicts with other vendors' names. Common practice is to use the vendor name followed by the chassis model name for the file name root.

This appendix does not fully describe the syntax of the chassis description file. The PXI specification is quite clear on that and has helpful examples. The key information included in the chassis description file is:

Chassis Description Contains general information about the chassis (the model name and vendor name). This also includes an indication of the number of instances of subsequent descriptions. For instance, the number of slots, and the number of trigger busses. This provides some redundancy in the file that helps detect syntax errors (for instance, if 8 slots are declared but only 7 are described).

Trigger Bus As with AXIe, PXI defines several trigger lines. Although disallowed by AXIe, PXI buses may connect a group of slots together into a *segment*. The chassis then provides buffering between segments that must be configured as to the direction of signal flow. This descriptor specifies the number of segments, and which slot is in each segment.

StarSystemTimingSet PXI Express defines additional differential star triggers. These are described by this descriptor and not used with AXIe.

StarTrigger This descriptor described which slots are connected to each PXI timing set (which in turn are related to a timing slot by the StarSystemTimingSet descriptor).

Slot The slot descriptor describes an individual slot. The only tags defined by PXI are the LocalBusLeft and LocalBusRight tags. For PXI, this specifies which slot is connected to the left or right local bus. For AXIe, the logical address is used.

PXI1BusSegment This descriptor provides information about the PXI1 (parallel) connections. There is a single occurrence of this descriptor for each parallel PCI bus in the chassis. The descriptor describes the PCI IDSEL number of each slot. Note that this is important for both conventional PXI1 slots and for slots that are hybrid.

A.8 Details of the PXI Chassis driver

PXI-6 also defines a chassis driver. This is not necessary for a pure PCIe system like AXIe.

The PXI chassis driver provides dynamic information about each chassis that is used along with static information provided in a chassis INI file. For parallel PXI, the chassis INI file identifies PCI IDSEL line associated with each slot. The chassis driver completes the information by providing the resource manager with the bus number of each PCI segment in the chassis.

The chassis driver follows a similar discovery algorithm as the system module driver, providing the resource manager with PCI bus number of the root bus in each chassis. The resource manager uses this along with information in the chassis .INI file and information returned by the peripheral driver to enumerate all PXI-1 peripherals and determine their slot number and bus number. This is very similar to the role of the system module driver since enumerating the system modules basically provides a list of the connected chassis as well.

The information provided by the chassis driver differs from what is provided by the system module driver in that the chassis driver provides a list of parallel PCI root buses.

A.9 Peripheral Module Description Files

PXI-4 specifies a module description file. This is an optional file that is not required or recommended for AXIe devices. The resource manager inserts the contents of the periperhal module description file in the slot descriptor of any slots containing the module type described by the module description file.

A.10 VISA PXI Plug-in Specification

The IVI 6.3 defines a standard VISA plug-in. This is a DLL that is dynamically loaded by VISA to do IO to PXI devices. Although not required by PXI or AXIe, this is a useful module to provide because it allows third party

VISA libraries to control arbitrary PXI modules. For some VISA vendors, this is required for the module to appear in their system console applications.

