PXI™-2

PXI Software Specification

PCI eXtensions for Instrumentation

An Implementation of CompactPCI™

Revision 2.4
October 18, 2012

PXI
Systems Alliance

IMPORTANT INFORMATION

Copyright

NOTICE

The *PXI Software Specification* is authored and copyrighted by the PXI Systems Alliance. The intent of the PXI Systems Alliance is for the *PXI Software Specification* to be an open industry standard supported by a wide variety of vendors and products. Vendors and users who are interested in developing PXI-compatible products or services, as well as parties who are interested in working with the PXI Systems Alliance to further promote PXI as an open industry standard, are invited to contact the PXI Systems Alliance for further information.

The PXI Systems Alliance wants to receive your comments on this specification. Visit the PXI Systems Alliance web site at `http://www.pxisa.org/` for contact information and to learn more about the PXI Systems Alliance.

The attention of adopters is directed to the possibility that compliance with or adoption of the PXI Systems Alliance specifications may require use of an invention covered by patent rights. The PXI Systems Alliance shall not be responsible for identifying patents for which a license may be required by any PXI Systems Alliance specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. PXI Systems Alliance specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

The information contained in this document is subject to change without notice. The material in this document details a PXI Systems Alliance specification in accordance with the license and notices set forth on this page. This document does not represent a commitment to implement any portion of this specification in any company's products.

The PXI Systems Alliance makes 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. The PXI Systems Alliance 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.

Compliance with this specification does not absolve manufacturers of PXI equipment from the requirements of safety and regulatory agencies (UL, CSA, FCC, IEC, etc.).

Trademarks

PXI™ is a trademarks of the PXI Systems Alliance.

PICMG™ and CompactPCI® are trademarks of the PCI Industrial Computation Manufacturers Group.

Product and company names are trademarks or trade names of their respective companies.

PXI Software Specification Revision History

This section is an overview of the revision history of the PXI Software Specification.

Revision 2.1, February 4, 2003

This is the first public revision of the PXI specification.

Revision 2.2, September 8, 2003

Added specification number and updated relevant documents list.

Revision 2.3, January 22, 2008

Added 64-bit Windows system framework. Adjusted references to VISA specifications. Corrected several errata.

Revision 2.4, October 18, 2012

Added new `chassis ini` file content related to the PXI Trigger Manager specified in PXI-9.

Added description of registration and selection mechanism for Resource Manager.

Modified description of Resource Manager algorithm to include assignment of Trigger Managers per chassis.

Added requirements for chassis description file names and the chassis description file path.

This Page Intentionally Left Blank

Contents

4. Service Registration and Configuration

Tables

1. Introduction

This section explains the objectives and scope of the PXI Software Specification. It also describes the intended audience and lists relevant terminology and documents. Note that this specification is intended to supplement the PXI Hardware Specification. Refer to the PXI Hardware Specification for general background on PXI and its electrical and mechanical requirements.

1.1 Objectives

The PXI software architecture, in addition to PXI's mechanical and electrical requirements, is a key component in furthering the standard's interoperability and ease of integration goals. The *PXI Software Specification* was created to supplement the *PXI Hardware Specification* in clarifying and addressing common software requirements in PXI systems. The software specification's purposes are to describe the capabilities of PXI hardware components using standard hardware description files and to promote interoperability among PXI vendors with respect to software requirements. The software specification addresses a variety of issues, including hardware description, hardware resource management, operating system framework definition, and the incorporation of existing instrumentation software standards.

The primary objective of the *PXI Software Specification* is to define a set of hardware description files for characterizing PXI components and their capabilities. Using standard file formats, device drivers, configuration software, and systems integrators can implement ease-of-use features such as geographic slot identification and chassis identification. These hardware description files can also serve as a repository for managing PXI hardware resources, including the PXI trigger bus, the PXI star trigger, and the PXI local bus.

A secondary objective of the *PXI Software Specification* is to define standard operating system frameworks and to incorporate existing instrumentation software standards. Additional software requirements include the support of standard operating system frameworks such as Microsoft Windows, and the support of the VISA instrumentation software standards maintained by the IVI Foundation.

1.2 Intended Audience and Scope

This specification is primarily intended for product developers interested in implementing and leveraging software features of the PXI platform. Hardware developers will be interested in using hardware description files for identifying and describing the capabilities of PXI hardware products such as chassis and system controller modules. Likewise, software developers and systems integrators should take advantage of hardware description files to manage PXI resources, including PXI triggers and the PXI local bus, and to implement features such as slot identification and chassis identification. Additionally, product developers and systems integrators should reference the operating system framework definitions to ensure system-level interoperability.

1.3 Background and Terminology

This section defines the acronyms and key words that are referred to throughout this specification. This specification uses the following acronyms:

- **API—**Application Programming Interface
- **CompactPCI** – PICMG 2.0 Specification
- **PCI—**Peripheral Component Interconnect; electrical specification defined by PCISIG
- **PCISIG—**PCI Special Interest Group
- **PICMG—**PCI Industrial Computer Manufacturers Group
- **PXI—**PCI eXtensions for Instrumentation
- **VISA—**Virtual Instrument Software Architecture
- **VPP—**VXI*plug&play* Specification maintained by the IVI Foundation

This specification uses several key words, which are defined as follows:

RULE: Rules SHALL be followed to ensure compatibility. A rule is characterized by the use of the words SHALL and SHALL NOT.

RECOMMENDATION: Recommendations consist of advice to implementers that will affect the usability of the final module. A recommendation is characterized by the use of the words SHOULD and SHOULD NOT.

PERMISSION: Permissions clarify the areas of the specification that are not specifically prohibited. Permissions reassure the reader that a certain approach is acceptable and will cause no problems. A permission is characterized by the use of the word MAY.

OBSERVATION: Observations spell out implications of rules and bring attention to things that might otherwise be overlooked. They also give the rationale behind certain rules, so that the reader understands why the rule must be followed.

MAY: A key word indicating flexibility of choice with no implied preference. This word is usually associated with a permission.

SHALL: A key word indicating a mandatory requirement. Designers SHALL implement such mandatory requirements to ensure interchangeability and to claim conformance with the specification. This word is usually associated with a rule.

SHOULD: A key word indicating flexibility of choice with a strongly preferred implementation. This word is usually associated with a recommendation.

1.4 Applicable Documents

This specification defines extensions to the base PCI and CompactPCI specifications referenced in this section. It is assumed that the reader has a thorough understanding of PCI and CompactPCI. The CompactPCI specification refers to several other applicable documents with which the reader may want to become familiar. This specification refers to the following documents directly:

- *PXI-1: PXI Hardware Specification*
- *VPP-4.3: The VISA Library Specification*
- *PXI-4: PXI Module Description File Specification*
- *PXI-6: PXI Express Software Specification*
- *PXI-9: PXI and PXI Express Trigger Management Specification*
- *PCI Local Bus Specification*
- *PICMG 2.0 R3.0 CompactPCI Specification*

2. Hardware Description Files

This section defines the formats of the hardware description files and describes their use.

2.1 Overview

The *PXI Hardware Specification* allows many variations of chassis and system controller modules. While many PCI hardware capabilities are self-describing (that is, their identities and capabilities can be determined using standard PCI hardware enumeration techniques), there is no standard hardware mechanism for identifying and managing many of the resources in a PXI system. The *PXI Software Specification* solves this problem by defining a set of hardware description files for PXI systems and the components that comprise them.

A primary goal of PXI's hardware description files is to enable application and device driver software to identify components based on their geographic characteristics (that is, chassis number and slot number) rather than their less user-friendly PCI logical address characteristics (PCI bus number, device number, and function number). For example, using a PXI system description file as a lookup table, an application or driver can map between a module's location on the PCI bus and its physical location in a PXI chassis. This functionality enables operators to quickly and easily distinguish between several similar modules using the chassis number and slot number.

Another goal of the hardware description files is to serve as a repository for managing PXI platform resources. PXI triggers, for example, are a shared hardware resource, and the trigger lines must be managed by a central reservation facility to guarantee the prevention of resource conflicts. Similarly, PXI's local bus lines are managed by software to guarantee that adjacent PXI modules do not use the local bus in a conflicting manner. In both of these cases, hardware description files serve as standard data storage for describing and managing these shared resources.

PXI hardware descriptions are contained in `.ini` files, which consist of ASCII text. The `.ini` file format is useful because it is both human readable and easily parsed by application and driver software.

The *PXI Software Specification* defines two hardware description file formats: system description files and chassis description files. The system description file is used to describe an overall PXI system and the components that comprise it. It is a collection of information obtained from several sources, including other hardware description files. The chassis description file is used to describe a PXI chassis and its features. Both of these file formats are described in detail below.

2.2 Common File Requirements

All hardware description files are `.ini` files. Each `.ini` file contains one or more sections, and each section contains one or more tag lines. Each tag line describes a specific property of the section.

In the context of PXI hardware description files, `.ini` file sections form *descriptors*. Descriptors describe PXI systems and the components that comprise them. Descriptors always correspond to unique `.ini` file sections.

RULE: Each `.ini` file SHALL contain only ASCII text and whitespace.

RULE: Horizontal whitespace SHALL be defined as any of the ASCII characters for horizontal tab and space.

RULE: Each `.ini` file SHALL contain only the following types of lines:
comment lines, blank lines, section headers, or tag lines.

RULE: A comment line SHALL begin with either the '#' character or the ';'character.

RULE: A section header line SHALL begin with the '[' character and end with the ']' character. Text between the two brackets SHALL identify the type of section.

RULE: ASCII double-quotes SHALL be the only valid characters used to quote values.

RULE: Scalar numeric values SHALL NOT be quoted.

RULE: List values, including lists of zero or one items, SHALL be quoted.

RULE: All scalar numeric values shall be radix-10 unless stated otherwise.

RULE: All values in the `.ini` files SHALL be considered case sensitive.

RULE: String values SHALL be quoted.

RULE: If at least one level of quotes are present surrounding a tag value, readers of the `.ini` file SHALL remove the outermost set of quotes before interpreting the value.

OBSERVATION: The above rule ensures backward compatibility with previous versions of this specification.

PERMISSION: Readers of `.ini` files MAY interpret a file in a case-insensitive manner to ensure compatibility with writers that predate this specification.

RULE: All readers of `.ini` files SHALL ignore sections with section headers that they do not recognize.

RULE: All readers of `.ini` files SHALL ignore lines that they do not recognize.

RULE: A tag line SHALL consist of the following three fields:
tag, '=' character, and value.

PERMISSION: The three fields in a tagline MAY be separated by any amount of horizontal whitespace.

.ini File Format Example

```
# This line is a comment.
[Section1]
ExampleTag = 1
IsSpecialSectionTag = "No"
ExampleListTag = "1,2,3,4"

[Section2]
ExampleTag = 2
IsSpecialSectionTag = "Yes"
ExampleListTag = "5,6,7,8"
```

2.2.1 Version Descriptor

PXI hardware description files include a version descriptor section. The version descriptor allows software to distinguish between `.ini` file formats as the *PXI Software Specification* evolves. For information on backward compatibility, refer to the *Backward Compatibility with Previous PXI Specifications* section below.

RULE: A hardware description file SHALL include a single version descriptor.

RULE: A version descriptor `.ini` section SHALL be named "Version".

RULE: Each version descriptor section SHALL contain one of each tag line type described in Table 2-1.

Table 2-1. Version Information Tag Line Descriptions

Tag	Valid Values	Description
Major	*x*, where *x* is a positive decimal integer.	This field indicates the major version number of a version *x.y*, where *x* is the major number and *y* is the minor number of the *PXI Software Specification* version that this file complies with.
Minor	*y*, where *y* is a positive decimal integer.	This field indicates the major version number of a version *x.y*, where *x* is the major number and *y* is the minor number of the *PXI Software Specification* version that this file complies with.

Version Descriptor Example

```
[Version]
Major = 2
Minor = 4
```

OBSERVATION: A version descriptor is useful for identifying the *PXI Software Specification* file format that a hardware description file complies with.

2.2.2 Backward Compatibility with Previous PXI Specifications

Backward compatibility is a key feature of the *PXI Software Specification*, and hardware descriptions files must be structured so that compatibility with previous PXI specification revisions can be achieved.

Beginning with the *PXI Software Specification*, Revision 2.1*,* the format of each type of hardware description file has been modified so that new features, such as multi-chassis PXI systems and multi-segmented PXI chassis, can be accurately described. To maintain backward compatibility, the following applies:

PERMISSION: In addition to the format defined in this specification, A PXI System Description file MAY include sections in the format of PXI Specifications prior to revision 2.1 of the *PXI Software Specification*.

Sections of the PXI System Description file that are in the format of specification revisions prior to version 2.1 are referred to as *legacy sections*.

OBSERVATION: None of the section headings in this specification overlap with headings defined in previous specifications. Multiple versions of the PXI System Description file format may exist together in the same System Description file.

RULE: Even if a PXI System Description file includes the legacy sections, it MUST include the sections defined in this specification.

OBSERVATION: For a System Description file to accurately describe a majority of PXI systems in use today, it must use the format defined in this specification. The legacy PXI System Description format cannot sufficiently describe modern PXI systems.

2.3 System Description Files

System description files describe PXI systems and their components. The system controller module and the one or more chassis that comprise a PXI system determine a system description. A system description enables a variety of software functionality, including geographic slot identification. Chassis description files, from which much of the system description content is derived, are discussed later in this chapter.

2.3.1 System Description Definitions

To develop a system description, it is useful to define descriptors for the following PXI system components:

- **Resource Manager** – A Resource Manager Descriptor provides information about the Resource Manager that created the system description file.
- **System** – A PXI System descriptor corresponds to a physical PXI system. A PXI System is a collection of chassis. Multiple chassis in a system are coupled in a software-transparent manner (that is, they are coupled via PCI-PCI bridging).
  - **Chassis** – A chassis descriptor corresponds to a physical PXI chassis in a system. Chassis can include PCI bus segments, trigger buses, trigger bridges, star triggers, and slots. Line mapping specifications may be used to identify chassis capabilities to software.
    - **PCI Bus Segments** – A PCI bus segment descriptor corresponds to a distinct, physical PCI bus in a chassis. PCI bus segments can contain slots, bridges, and other backplane devices. Multiple PCI bus segments are linked within a chassis using PCI-PCI bridging.
    - **Trigger Buses** – A PXI trigger bus descriptor corresponds to a physical trigger bus in a chassis. A trigger bus is characterized by a list of slots sharing the physical trigger bus connection. Chassis can contain multiple trigger buses.
    - **Trigger Bridges** – A PXI trigger bridge descriptor corresponds to a physical trigger bridge in a PXI chassis. Each trigger bridge descriptor represents the possible unidirectional routes that can be established between two buses; if a physical trigger bridge can be used to establish routes in either direction between these buses, two trigger bridge descriptors must represent it, one for each direction. A chassis may contain multiple trigger bridges.
    - **Line Mapping Specifications** – A line mapping specification does not represent a physical chassis component, but sets out the possible routes that a trigger bridge can establish between two adjacent trigger buses. This line mapping provides software with detailed information about the routing capabilities that the chassis supports. These routes can be established through calls made to the chassis Trigger Manager, as described in *PXI-9: PXI and PXI Express Trigger Management Specification*. Multiple line mappings can describe a chassis' routing capabilities.
    - **Star Triggers –** A PXI star trigger descriptor corresponds to a physical set of star triggers in a chassis. A set of star triggers is characterized by a star trigger controller slot number and a mapping of PXI_STAR lines (defined in the *PXI Hardware Specification*) to peripheral slot numbers. A chassis can contain multiple sets of star triggers.
    - **Slots** –A PXI slot descriptor corresponds to a physical slot in a chassis. A slot is characterized by a geographic address, a PCI logical address, local bus routings, and other special capabilities. A chassis has multiple slots.

In addition, a *Resource Manager* is defined as the entity responsible for creating a PXI system description file. For example, the responsibilities of a Resource Manager might be accomplished by a systems integrator, or a software utility might be provided to automate the Resource Manager algorithm.

RULE: A system controller module manufacturer SHALL provide either a system description file for each supported system configuration or a Resource Manager utility that can manage the system description file.

OBSERVATION: A *system controller module* is any module that resides in slot 1 of a PXI chassis.

RECOMMENDATION: A system controller module manufacturer SHOULD provide a utility that can automate the Resource Manager algorithm.

OBSERVATION: *PXI-1: PXI Hardware Specification* does not define hardware mechanisms that provide for the automatic discovery of PXI chassis or PXI system controller modules. As a result, a software resource manager may require input from the user to facilitate discovery of these components.

RULE: A system description file SHALL be named `pxisys.ini`. The `pxisys.ini` file SHALL be located in the `<windows>` directory (for example, `c:\windows or c:\winnt`).

RECOMMENDATION: To aid systems integrators and operators, PXI module configuration and driver software SHOULD use geographic addressing information, available in a PXI system description file, to present chassis and slot locations for PXI modules via a user interface.

2.3.2 Resource Manager Descriptor

The Resource Manager descriptor provides information about how the system description file was generated. This information is intended for debugging purposes.

RULE: A system description file SHALL contain one and only one Resource Manager descriptor.

RULE: The Resource Manager descriptor `.ini` section header SHALL be named *ResourceManager*.

RULE: The Resource Manager descriptor section SHALL contain one of each tag line type described in Table 2-4.

Table 2-2. Version Information Tag Line Descriptions

Tag	Valid Values	Description
Name	The name of the Resource Manager.	This field identifies the Resource Manager that last edited or created the System Description File.
Version	A vendor-defined string.	This field should be populated with a vendor-defined string describing the version of the Resource Manager that generated the System Description File.
Timestamp	A string containing a human-readable date and time.	This field indicates the date and time when the file was last written by the Resource Manager.

Resource Manager Descriptor Example

```
[ResourceManager]
Name = "PXISA Resource Manager"
Version = "1.0.0"
Timestamp = "August 29, 2011, 02:00:00 PM GMT-0400"
```

RULE: A software Resource Manager SHALL use its own name, as displayed in the Resource Manager's category key of the services tree, as the value for the Vendor tag.

RECOMMENDATION: If the system description file is generated by any means other than by a software Resource Manager, the name tag SHOULD describe the entity creating the file.

RECOMMENDATION: A software Resource Manager SHOULD populate the Version tag with a meaningful string that is unique to each version of the Resource Manager made available by the specified vendor.

PERMISSION: If the system description file is generated by any means other than by a software Resource Manager, any value, including an empty string, MAY be used for the Version tags.

RULE: The value of the Timestamp tag SHALL be a human readable date and time, with a resolution of at least 1 second.

RECOMMENDATION: The value of the Timestamp tag SHOULD include a time zone indication that is the same as the time zone of the system on which the system description file was created.

OBSERVATION: The Timestamp tag is intended to be used for debugging purposes only. For a date and time suitable for interpretation by software, the operating system provides a more relevant and accessible mechanism to retrieve the time when the system description file was last modified.

2.3.3 System Descriptor

The system descriptor contains highest-level information about a PXI system. PXI systems are characterized by the chassis that comprise the system, and the system descriptor contains a list of these chassis.

RULE: A system description file SHALL contain one and only one system descriptor.

RULE: The system descriptor `.ini` section header SHALL be named “System”.

RULE: Each system descriptor section SHALL contain one of each tag line types described in Table 2-3.

Table 2-3. System Description File – System Tag Line Descriptions

Tag	Valid Values	Description
Chassis List	A comma-separated list of *n*, where *n* is a decimal integer such that *n* >= 1.	This tag enumerates the chassis in a PXI system.

System Descriptor Example

```
# This section describes a PXI system with two chassis.
[System]
ChassisList = "1,2"
```

RULE: PXI chassis, specified with the ChassisList tag, SHALL be enumerated by a Resource Manager when collecting information regarding each chassis in the PXI system.

OBSERVATION: A Resource Manager can enumerate chassis using a variety of mechanisms. For example, a Resource Manager utility can present a user interface, allowing a user to identify the types of chassis included in the system.

RULE: Multiple chassis SHALL be uniquely numbered in the ChassisList tag.

OBSERVATION: Chassis can be numbered in an arbitrary fashion. For example, chassis can be numbered according to their order of discovery using a depth-first PCI traversal algorithm.

2.3.4 Chassis Descriptor

A chassis descriptor provides a high-level description of an individual PXI chassis in a system. A chassis descriptor contains collections of the components that comprise a chassis, including PCI bus segments, trigger buses, sets of star triggers, and slots.

RULE: A system description file SHALL contain a distinct chassis descriptor for each physical chassis that comprises the PXI system.

OBSERVATION: Chassis are enumerated using a system descriptor's ChassisList tag.

RULE: A chassis descriptor SHALL be named "Chassis*N*", where *N* is the chassis number.

RULE: A Resource Manager SHALL derive chassis numbers from the ChassisList tag of a system descriptor (see Table 2-3).

RECOMMENDATION: The chassis number SHOULD be physically viewable on a chassis to assist operators in locating peripheral modules.

RULE: Each chassis descriptor SHALL contain one of each of the tag line types described in Table 2-4.

Table 2-4. System Description File – Chassis Tag Line Description

Tag	Valid Values	Description
PCIBusSegmentList	A comma-separated list of *n*, where *n* is a decimal integer such that 1 <= *n* <= 255.	This tag enumerates the PCI bus segments in a chassis.
TriggerBusList	A comma-separated list of *n*, where *n* is a decimal integer such that *n* >= 1.	This tag enumerates the trigger buses in a chassis.
StarTriggerList	A comma-separated list of *n*, where *n* is a decimal integer such that *n* >= 1.	This tag enumerates the sets of star triggers in a chassis.
TriggerBridgeList	A comma-separated list of *n*, where *n* is a decimal integer such that *n* >= 1.	This tag enumerates the trigger bridges in a chassis.
LineMappingSpecList	A comma-separated list of *n*, where *n* is a decimal integer such that *n* >= 1.	This tag enumerates the line mapping specifications that exist for a chassis.
SlotList	A comma-separated list of *n*, where *n* is a decimal integer such that *n* >= 1.	This tag enumerates the slots in a chassis.
TriggerManager	A string indicating the relative path to the Trigger Manager to use for the chassis, based at the root of the Trigger Managers portion of the Services Tree, or "None."	This tag provides an indirect reference to the Trigger Manager for the chassis.
DescriptionFile	A string containing a filename.	This tag identifies the filename of the chassis description file.

Table 2-4. System Description File – Chassis Tag Line Description (Continued)

Tag	Valid Values	Description
Model	A string indicating the model name for this chassis.	This tag identifies a chassis model name.
Vendor	A string indicating the vendor name for this chassis.	This tag identifies a chassis vendor name.

Chassis Descriptor Example

```
# This example describes a 3-segment, 18-slot PXI chassis
# with 2 bidirectional trigger bridges that have equivalent
# routing capabilities
[Chassis1]
PCIBusSegmentList = "1,2,3"
TriggerBusList = "1,2,3"
TriggerBridgeList = "1,2,3,4"
LineMappingSpecList = "1"
StarTriggerList = "1"
SlotList = "1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18"
TriggerManager = "PXISA\Example 18-Slot Chassis"
DescriptionFile = "PXISA Example 18-Slot Chassis.ini"
Model = “Example 18-Slot Chassis”
Vendor = “PXISA”
```

RULE: With the exceptions of the “TriggerManager” and “DescriptionFile” tags, a Resource Manager SHALL derive the tag values in Table 2-4 from the tag values of the corresponding chassis description file’s chassis descriptor (see Table 2-11).

RULE: A Resource Manager SHALL set the “DescriptionFile” tag to the filename of the chassis description file for the chassis.

OBSERVATION: Software can use the “DescriptionFile” tag to easily locate the chassis description file for the chassis. This is useful if the vendor has included additional information in the chassis description file that has not been copied into the system description file, but which may be useful for vendor-specific behaviors.

OBSERVATION: The “DescriptionFile” tag does not include the full path to the chassis description file, because all chassis description files are in the same directory. Refer to section 2.4.

RULE: A Resource Manager SHALL set the TriggerBridgeList and LineMappingSpecList tag values to an empty list if the corresponding chassis description file’s chassis descriptor does not contain these tags.

RULE: A Resource Manager SHALL set the value of the TriggerManager tag to *Vendor\Model*, where *Vendor* is the chassis vendor and *Model* is the chassis model, to indicate the specific Trigger Manager specified for the chassis model in the Services Tree, if such a specification is available there.

RULE: For chassis that do not have a specific trigger manager indicated in the Services Tree, a Resource Manager SHALL set the value of the TriggerManager tag to *Vendor*, where *Vendor* is the chassis vendor, to indicate the vendor default Trigger Manager for the vendor of the chassis, if such a vendor default trigger manager is specified.

RULE: For chassis that do not have a corresponding model-specific or vendor default Trigger Manager specified in the Services Tree, a Resource Manager SHALL set the value of the TriggerManager tag to *Vendor*, where *Vendor* is the default trigger manager’s vendor.

RULE: A Resource Manager SHALL set the TriggerManager field to “None” if there is no chassis-appropriate trigger manager available, and the default Trigger Manager (refer to section 4.3.2) is set to “None.”

OBSERVATION: If the TriggerManager field is “None,” it signifies that no Trigger Manager is available for the chassis on the system, and any attempt to use a Trigger Manager for the chassis will fail.

OBSERVATION: This use case is expected to occur most commonly during the transition to the new specifications and become less common with time. The ability to use a default Trigger Manager from any vendor is intended to further limit the scope of this case.

OBSERVATION: The value for the TriggerManager tag is the relative path from the Trigger Managers category key to the key containing the trigger manager’s specification, with elements separated by a backslash (“\”).

OBSERVATION: If the Services Tree does not specify a Trigger Manager for a chassis or for the vendor of that chassis, the software for the chassis predates *PXI-9: PXI and PXI Express Trigger Management Specification*. In this case, a default Trigger Manager is selected to handle reservations for the chassis, so that the chassis can work with software designed to call the Trigger Manager APIs. Refer to section 4.3.2 for more information about the default Trigger Manager.

2.3.5 PCI Bus Segment Descriptor

A PCI bus segment descriptor describes an individual PCI bus segment in a chassis.

RULE: A system description file SHALL contain a distinct PCI bus segment descriptor for each physical PCI bus segment in the system.

RULE: A PCI bus segment descriptor SHALL be named “Chassis*M*PCIBusSegment*N*”, where *M* is the chassis number, and *N* is the PCI bus segment number.

RULE: A Resource Manager SHALL derive PCI bus segment numbers from the PCIBusSegmentList tag of the corresponding chassis descriptor (see Table 2-4).

OBSERVATION: While each PCI bus segment number will uniquely correspond to a PCI bus number, the PCI bus segment number will not necessarily be equal to the corresponding PCI bus number.

RULE: Each PCI bus segment descriptor SHALL contain one of each of tag line type described in Table 2-5.

Table 2-5. System Description File – PCI Bus Segment Tag Line Descriptions

Tag	Valid Values	Description
SlotList	A comma-separated list of *n*, where *n* is a decimal integer such that *n* >= 1.	This tag enumerates the physical slots on a PCI bus segment.

PCI Bus Segment Descriptor Example

```
# This example describes the third bus segment of
# an 18-slot PXI chassis
[Chassis1PCIBusSegment3]
SlotList = "13,14,15,16,17,18"
```

RULE: A Resource Manager SHALL derive the tag values in Table 2-5 from the tag values of the corresponding chassis description file’s PCI Bus Segment descriptor (see Table 2-12).

PERMISSION: A PCI bus segment descriptor that describes a segment with no PXI slots will contain an empty slot list. In this case, the PCI bus segment descriptor MAY be excluded from the system description file.

2.3.6 Trigger Bus Descriptor

A trigger bus descriptor describes an individual trigger bus in a PXI chassis. A trigger bus is characterized by a list of slots that reside on the trigger bus.

RULE: A system description file SHALL contain a distinct PXI trigger bus descriptor for each physical PXI trigger bus in the system.

RULE: A trigger bus descriptor SHALL be named "Chassis*M*TriggerBus*N*", where *M* is the chassis number and *N* is the trigger bus number.

RULE: A Resource Manager SHALL derive trigger bus numbers from the TriggerBusList tag of the corresponding chassis descriptor (see Table 2-4).

OBSERVATION: While each trigger bus number will uniquely correspond to a set of PXI slots, there is not necessarily a one-to-one correspondence between trigger buses and PCI bus segments.

RULE: Each trigger bus descriptor SHALL contain one of each of the tag line types described in Table 2-6.

Table 2-6. System Description File - Trigger Bus Tag Line Descriptions

Tag	Valid Values	Description
SlotList	A comma-separated list of *n*, where *n* is a decimal integer such that *n* >= 1.	This tag enumerates the slots on a trigger bus.

Trigger Bus Descriptor Example

```
# This example describes the first trigger bus of a
# 3-segment, 18-slot chassis.
[Chassis1TriggerBus1]
SlotList = "1,2,3,4,5,6"
```

RULE: A Resource Manager SHALL derive the tag values in Table 2-6 from the tag values of the corresponding chassis description file's Trigger Bus descriptor (see Table 2-13).

2.3.7 Trigger Bridge Descriptor

A trigger bridge descriptor describes a unidirectional trigger bridge in a PXI chassis.

RULE: A trigger bridge descriptor SHALL be named *ChassisMTriggerBridgeN*, where *M* is the chassis number and *N* is the number of the trigger bridge.

RULE: A Resource Manager SHALL derive trigger bridge descriptor numbers from the TriggerBridgeList tag of the chassis descriptor (refer to Table 2-4).

RULE: Each trigger bridge descriptor SHALL contain one of each of the tagline types described in Table 2-7.

Table 2-7. System Description File – Trigger Bridge Descriptions

Tag	Valid Values	Description
SourceTriggerBus	*n*, where *n* is a decimal integer such that *n* >= 1.	The source trigger bus for this trigger bridge.
DestinationTriggerBus	*n*, where *n* is a decimal integer such that *n* >= 1.	The destination trigger bus for this trigger bridge.
LineMappingSpec	*n*, where *n* is a decimal integer such that *n* >= 1.	The number of the line mapping spec that describes the routing capabilities of this trigger bridge.

Trigger Bridge Descriptor Example

```
# This example describes a trigger bridge that
# can route signals from trigger bus 1 to trigger
# bus 2, with line-by-line capabilities described by
# LineMappingSpec 1.
[Chassis1TriggerBridge1]
SourceTriggerBus = 1
DestinationTriggerBus = 2
LineMappingSpec = 1
```

RULE: A Resource Manager SHALL derive the tag values in Table 2-7 from the tag values of the corresponding chassis description file's Trigger Bridge descriptor.

2.3.8 Line Mapping Specification Descriptor

A line mapping specification describes the possible routes that can be established between a given source bus and destination bus. The line mapping specification is in a separate descriptor from the trigger bridge so that it can be referenced from multiple trigger bridge descriptors, avoiding unnecessary duplication of information about routing capabilities.

OBSERVATION: There is no direct relationship between the number of physical trigger bridges in a chassis and the number of line mapping specification descriptors necessary; there should be as many line mapping spec descriptors as there are unique sets of bus-to-bus routing capabilities provided by trigger routers. For example, if a chassis has three trigger bridges with equivalent routing capabilities in each direction, then only a single line mapping specification descriptor is necessary.

RULE: A line mapping specification descriptor SHALL be named *ChassisMLineMappingSpecN,* where *M* is the chassis number and *N* is the number for the line mapping specification.

RULE: A Resource Manager SHALL derive line mapping spec descriptor numbers from the LineMappingSpecList tag of the chassis descriptor (refer to Table 2-4).

RULE: Each line mapping spec descriptor SHALL contain one of each of the tagline types described in Table 2-8.

Table 2-8. System Description File – Line Mapping Spec Descriptions

Tag	Valid Values	Description
PXI_TRIG*n*, where *n* is an integer that represents a PXI trigger line on the source trigger bus of the trigger bridge referencing this descriptor. One tag must exist for each trigger line on the source bus.	A comma-separated list of *n*, where *n* is a decimal integer such that 0 <= *n* <= 7.	This tag enumerates the lines on the destination trigger bus to which the signal on the source line can be routed.

Line Mapping Spec Descriptor Example

```
# This example describes a line mapping in which
# the referencing trigger bridge can map any line on the
# source trigger bus to any line on the destination
# trigger bus.
[Chassis1LineMappingSpec1]
PXI_TRIG0 = "0,1,2,3,4,5,6,7"
PXI_TRIG1 = "0,1,2,3,4,5,6,7"
PXI_TRIG2 = "0,1,2,3,4,5,6,7"
PXI_TRIG3 = "0,1,2,3,4,5,6,7"
PXI_TRIG4 = "0,1,2,3,4,5,6,7"
PXI_TRIG5 = "0,1,2,3,4,5,6,7"
PXI_TRIG6 = "0,1,2,3,4,5,6,7"
PXI_TRIG7 = "0,1,2,3,4,5,6,7"
```

RULE: A Resource Manager SHALL derive the tag values in Table 2-8 from the tag values of the corresponding chassis description file's Line Mapping Spec descriptor (refer to Table 2-8).

2.3.9 Star Trigger Descriptor

A star trigger descriptor describes an individual set of star triggers in a PXI chassis. A star trigger descriptor is characterized by a star trigger controller slot number and a mapping of PXI_STAR lines, as defined in the *PXI Hardware Specification*, to peripheral slot numbers.

RULE: A system description file SHALL contain a distinct PXI star trigger descriptor for each physical set of PXI star triggers in the system.

RULE: A trigger bus descriptor SHALL be named "Chassis*M*StarTrigger*N*", where *M* is the chassis number and *N* is the number for the set of star triggers.

RULE: A Resource Manager SHALL derive star trigger descriptor numbers from the StarTriggerList tag of the corresponding chassis descriptor (see Table 2-4).

RULE: Each star trigger descriptor SHALL contain one of each of the tag line types described in Table 2-9.

Table 2-9. System Description File – Star Trigger Tag Line Descriptions

Tag	Valid Values	Description
ControllerSlot	A decimal integer *n*, where *n* is a decimal integer such that *n* >= 1.	This tag specifies the star trigger controller slot number for a PXI_STAR lines in a set of star triggers.
PXI_STAR*n* (where *n* is a decimal integer such that 0 <= *n* <= 12), for each PXI star trigger line physically routed to a PXI slot	A decimal integer *m*, where *m* is a valid PXI slot number that connects to the star trigger line.	This tag specifies the PXI_STAR line to slot mapping for a set of star triggers.

Star Trigger Descriptor Example

```
# This example describes a set of star triggers for a
# 3-segment, 18-slot chassis.
[Chassis1StarTrigger1]
ControllerSlot = 2
PXI_STAR0 = 3
PXI_STAR1 = 4
PXI_STAR2 = 5
PXI_STAR3 = 6
PXI_STAR4 = 7
PXI_STAR5 = 8
PXI_STAR6 = 9
PXI_STAR7 = 10
PXI_STAR8 = 11
PXI_STAR9 = 12
PXI_STAR10 = 13
PXI_STAR11 = 14
PXI_STAR12 = 15
```

RULE: A Resource Manager SHALL derive the tag values in Table 2-9 from the tag values of the corresponding chassis description file's Star Trigger descriptor (see Table 2-16).

OBSERVATION: The star trigger descriptor allows configuration software to describe alternative star trigger line mappings.

OBSERVATION: If a star trigger line is not routed to a PXI slot, the corresponding PXI_STAR*n* tag will not be listed in the star trigger bus descriptor.

2.3.10 Slot Descriptor

A slot descriptor describes an individual slot in a chassis. A slot descriptor is characterized by the features of the slot it describes, including routing information for the slot's local bus lines and the PCI logical address for a module that might occupy the slot. The slot descriptor serves as a lookup facility for applications and driver software interested in geographic slot identification.

RULE: A system description file SHALL contain a distinct slot descriptor for each physical slot in the PXI system.

RULE: A slot descriptor SHALL be named "Chassis*M*Slot*N*", where *M* is the chassis number, and *N* is the physical slot number.

RULE: A Resource Manager SHALL derive slot numbers from the SlotList tag of the corresponding chassis descriptor (see Table 2-4).

RULE: Each slot descriptor SHALL contain one of each of non-shaded tag line type described in Table 2-10.

PERMISSION: Application and device driver software MAY continue to use the shaded fields of Table 2-10. These fields may be removed in a future revision.

RECOMMENDATION: New software development SHOULD use the non-shaded fields.

Table 2-10. System Description File – Slot Tag Line Descriptions

Tag	Valid Values	Description
PCISlotPath	A string containing a comma-separated list of *n*, where *n* is a hexadecimal integer indicating the slot path of this PXI slot.	This tag indicates the PCI slot path for a slot.
PCISlotPathRootBus	*n*, where *n* is a decimal integer such that 0 <= *n* <= 255.	This tag indicates the bus number of the PCI root at which the PCISlotPath is based.
LocalBusLeft	A valid slot descriptor. A valid star trigger descriptor. (Other).	This tag indicates how a slot routes its local bus pins to the left.
LocalBusRight	A valid slot descriptor. (Other).	This tag indicates how a slot routes its local bus pins to the right.
PCIBusNumber	*n*, where *n* is a decimal integer such that 0 <= *n* <= 255.	This tag indicates the PCI bus number for a slot.
PCIDeviceNumber	*n*, where *n* is a decimal integer such that 0 <= *n* <= 31.	This tag indicates the PCI device number for a slot.
ExternalBackplaneInterface	None. (Other).	If a slot routes to an external backplane interface, this tag specifies the name of that interface.

Slot Descriptor Example

```
# This example describes Slot 2 of an 8-slot PXI chassis.
[Chassis1Slot2]
# To calculate the slot path, we note that this chassis sits
# behind a PCI-PCI bridge residing on PCI bus 0 at PCI
# device 17
PCISlotPath = "98,88"
PCISlotPathRootBus = 0
PCIBusNumber = 2
PCIDeviceNumber = 19
LocalBusLeft = "StarTrigger1"
LocalBusRight = "Chassis1Slot3"
```

RULE: A Resource Manager SHALL derive the LocalBusLeft, LocalBusRight, and ExternalBackplaneInterface tag values from the tag values in the corresponding chassis description file's slot descriptor (see Table 2-18).

RULE: A PXI Resource Manager SHALL derive the PCISlotPath, PCISlotPathRootBus, PCIBusNumber, and PCIDeviceNumber tag values from the controller IDSEL routing information, the PCI bus segment IDSEL routing information (see Table 2-12), and the PCI bus hierarchy.

OBSERVATION: A PXI slot that does not implement the full set of PXI features, such as a CompactPCI-only slot, will have tag values corresponding to PXI features set to "None". For example, a CompactPCI-only slot descriptor would have LocalBusLeft and LocalBusRight tags set to "None". In addition, this slot would not be present in the SlotList for a trigger bus, and it would not belong to a set of star triggers.

2.3.10.1 PCI Slot Path

To facilitate geographic slot identification, it is useful to introduce the concept of a PCI slot path. The purpose of a PCI slot path is to describe the PCI bus hierarchy in a manner independent of the PCI bus number. PCI slot paths are a sequence of hexadecimal values representing the PCI device number and function number of a PCI module and each parent PCI bridge that routes the module to the host PCI bridge (bus 0). Each byte of a slot path corresponds to the PCI BIOS device/function number encoding for the current bridge in the path. The encoding is calculated as follows:

PCI Slot Path Byte = (PCI Device Number << 3) | PCI Function Number

PCI Slot Path Example

Consider a PXI slot located on PCI bus #2, device #17d. This slot is subordinate to a PCI-PCI bridge with primary bus #0, device #14d. The slot path for this device would be "88,70", that is, ((17d << 3) | 0) followed by ((14d << 3) | 0).

OBSERVATION: Slot paths are useful because they change less frequently than PCI bus numbers. In PXI systems, if a PCI bus number changes, the PCI slot path for slots on that segment do not necessarily change.

OBSERVATION: A slot path describes the hierarchical location of a device relative to a PCI root. Because some systems have more than one PCI root, this specification requires that wherever a slot path is indicated in a description file, it is accompanied by the bus number of the root to which it applies.

RECOMMENDATION: Slot paths, along with their relevant root bus, SHOULD be used for mapping between PCI logical addresses to PXI geographic addresses.

RULE: Slot paths SHALL be represented as strings in `.ini` files.

2.3.10.2 Local Bus Routings

The slot descriptor provides a means for specifying how the local bus lines are routed for a given slot.

OBSERVATION: The LocalBusLeft and LocalBusRight tags will usually specify the slot descriptor for the slot to the left and right of the current slot, respectively.

OBSERVATION: If a slot does not route its local bus pins (left or right) to a neighboring slot's local bus, the LocalBus tags can be used to specify a special slot capability.

OBSERVATION: The LocalBusLeft tag for PXI slot 2 can be used to specify the Star Trigger capability, pointing to a chassis' StarTrigger descriptor.

OBSERVATION: The LocalBusRight tag to the rightmost slot in a chassis can be used to specify a connection to an external backplane interface.

RECOMMENDATION: The LocalBusLeft and LocalBusRight tags SHOULD be used to specify an external backplane connection. The ExternalBackplaneInterface tag is provided for backward compatibility only.

2.3.11 System Description File Example

```
# This example describes a PXI system with two chassis.
# The first chassis (Chassis1) has a single PXI bus
# segment and 8 PXI slots, described by the chassis
# description file in Example 2.4.10.1.
# The second chassis (Chassis2) has 3 PXI bus
# segments and 18 PXI slots, described by the chassis
# description file in Example 2.4.10.2.

# Assumptions:
# The two chassis (Chassis1 and Chassis2) are linked
# together using PXI-PXI bridging. The first chassis
# in the daisy-chain (Chassis1) contains a system
# controller module with a PCI-PCI bridge at PCI bus #0,
# device #30, function #0. Its corresponding PCI slot
# path node is 0xF0. This bridge forms the PXI bus
# segment (PCI bus #1) for Chassis1. The PXI-PXI bridge
# resides in slot #5 (PCI bus #1, device #12, function #0)
# of Chassis1. This split-bridge forms the first PXI
# segment of Chassis2 (PCI bus #3). Its corresponding
# PCI slot path node is 0x60. Chassis2 contains three
# PXI bus segments. The first segment contains a PCI-PCI
# bridge at PCI bus #3 device #12, function #0. Its
# corresponding PCI slot path node is 0x60. This
# bridge forms the second PXI bus segment (PCI bus #4).
# The second segment contains a PCI-PCI bridge at
# PCI bus #4, device #12, function #0. Its corresponding
# PCI slot path node is 0x60. This bridge forms the third
# PXI bus segment. The first chassis has no trigger routing
# capabilities. The second chassis has a
# trigger bridge between each segment. Any trigger
# line in the first segment can be routed to any line
# in the second segment, and vice-versa. Also, any line in
# the second segment can be routed to the same line in
# the third segment.

[Version]
Major = 2
Minor = 4

[PXI System]
ChassisList = "1,2"

[Chassis1]
Model = "Example 8-Slot Chassis"
Vendor = "PXISA"
```

```
PCIBusSegmentList = "1"
SlotList = "1,2,3,4,5,6,7,8"
TriggerBusList = "1"
TriggerManager = "PXISA"
StarTriggerList = "1"

[Chassis1StarTrigger1]
ControllerSlot = 2
PXI_STAR0 = 3
PXI_STAR1 = 4
PXI_STAR2 = 5
PXI_STAR3 = 6
PXI_STAR4 = 7
PXI_STAR5 = 8

[Chassis1PCIBusSegment1]
SlotList = "1,2,3,4,5,6,7,8"

[Chassis1TriggerBus1]
SlotList = "1,2,3,4,5,6,7,8"

[Chassis1Slot1]
PCISlotPath = "None"
PCISlotPathRootBus = "None"
PCIBusNumber = "None"
PCIDeviceNumber = "None"
LocalBusLeft = "None"
LocalBusRight = "None"
ExternalBackplaneInterface = "None"

[Chassis1Slot2]
PCISlotPath = "78,F0"
PCISlotPathRootBus = 0
PCIBusNumber = 1
PCIDeviceNumber = 15
LocalBusLeft = "StarTrigger1"
LocalBusRight = "Slot3"
ExternalBackplaneInterface = "None"

[Chassis1Slot3]
PCISlotPath = "70,F0"
PCISlotPathRootBus = 0
PCIBusNumber = 1
PCIDeviceNumber = 14
LocalBusLeft = "Slot2"
LocalBusRight = "Slot4"
ExternalBackplaneInterface = "None"

[Chassis1Slot4]
PCISlotPath = "68,F0"
PCISlotPathRootBus = 0
PCIBusNumber = 1
PCIDeviceNumber = 13
LocalBusLeft = "Slot3"
```

```
LocalBusRight = "Slot5"
ExternalBackplaneInterface = "None"

[Chassis1Slot5]
PCISlotPath = "60,F0"
PCISlotPathRootBus = 0
PCIBusNumber = 1
PCIDeviceNumber = 12
LocalBusLeft = "Slot4"
LocalBusRight = "Slot6"
ExternalBackplaneInterface = "None"

[Chassis1Slot6]
PCISlotPath = "58,F0"
PCISlotPathRootBus = 0
PCIBusNumber = 1
PCIDeviceNumber = 11
LocalBusLeft = "Slot5"
LocalBusRight = "Slot7"
ExternalBackplaneInterface = "None"

[Chassis1Slot7]
PCISlotPath = "50,F0"
PCISlotPathRootBus = 0
PCIBusNumber = 1
PCIDeviceNumber = 10
LocalBusLeft = "Slot6"
LocalBusRight = "Slot8"
ExternalBackplaneInterface = "None"

[Chassis1Slot8]
PCISlotPath = "48,F0"
PCISlotPathRootBus = 0
PCIBusNumber = 1
PCIDeviceNumber = 9
LocalBusLeft = "Slot7"
LocalBusRight = "None"
ExternalBackplaneInterface = "None"

[Chassis2]
Model = "Example 18-Slot Chassis"
Vendor = "PXISA"
PCIBusSegmentList = "1,2,3"
SlotList = "1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18"
TriggerBusList = "1,2,3"
TriggerBridgeList = "1,2,3"
LineMappingSpecList = "1,2"
TriggerManager = "PXISA\Example 18-Slot Chassis"
StarTriggerList = "1"

[Chassis2TriggerBridge1]
SourceTriggerBus = 1
DestinationTriggerBus = 2
LineMappingSpec = 1
```

```
[Chassis2TriggerBridge2]
SourceTriggerBus = 2
DestinationTriggerBus = 1
LineMappingSpec = 1

[Chassis2TriggerBridge3]
SourceTriggerBus = 2
DestinationTriggerBus = 3
LineMappingSpec = 2

[Chassis2LineMappingSpec1]
PXI_TRIG0 = "0,1,2,3,4,5,6,7"
PXI_TRIG1 = "0,1,2,3,4,5,6,7"
PXI_TRIG2 = "0,1,2,3,4,5,6,7"
PXI_TRIG3 = "0,1,2,3,4,5,6,7"
PXI_TRIG4 = "0,1,2,3,4,5,6,7"
PXI_TRIG5 = "0,1,2,3,4,5,6,7"
PXI_TRIG6 = "0,1,2,3,4,5,6,7"
PXI_TRIG7 = "0,1,2,3,4,5,6,7"

[Chassis2LineMappingSpec2]
PXI_TRIG0 = "0"
PXI_TRIG1 = "1"
PXI_TRIG2 = "2"
PXI_TRIG3 = "3"
PXI_TRIG4 = "4"
PXI_TRIG5 = "5"
PXI_TRIG6 = "6"
PXI_TRIG7 = "7"

[Chassis2StarTrigger1]
ControllerSlot = 2
PXI_STAR0 = 3
PXI_STAR1 = 4
PXI_STAR2 = 5
PXI_STAR3 = 6
PXI_STAR4 = 7
PXI_STAR5 = 8
PXI_STAR6 = 9
PXI_STAR7 = 10
PXI_STAR8 = 11
PXI_STAR9 = 12
PXI_STAR10 = 13
PXI_STAR11 = 14
PXI_STAR12 = 15

[Chassis2PCIBusSegment1]
SlotList = "1,2,3,4,5,6"

[Chassis2TriggerBus1]
SlotList = "1,2,3,4,5,6"
```

```
[Chassis2Slot1]
PCISlotPath = "None"
PCISlotPathRootBus = "None"
PCIBusNumber = "None"
PCIDeviceNumber = "None"
LocalBusLeft = "None"
LocalBusRight = "None"
ExternalBackplaneInterface = "None"

[Chassis2Slot2]
PCISlotPath = "78,60,F0"
PCISlotPathRootBus = 0
PCIBusNumber = 3
PCIDeviceNumber = 15
LocalBusLeft = "StarTrigger1"
LocalBusRight = "Slot3"
ExternalBackplaneInterface = "None"

[Chassis2Slot3]
PCISlotPath = "70,60,F0"
PCISlotPathRootBus = 0
PCIBusNumber = 3
PCIDeviceNumber = 14
LocalBusLeft = "Slot2"
LocalBusRight = "Slot4"
ExternalBackplaneInterface = "None"

[Chassis2Slot4]
PCISlotPath = "68,60,F0"
PCISlotPathRootBus = 0
PCIBusNumber = 3
PCIDeviceNumber = 13
LocalBusLeft = "Slot3"
LocalBusRight = "Slot5"
ExternalBackplaneInterface = "None"

[Chassis2Slot5]
PCISlotPath = "58,60,F0"
PCISlotPathRootBus = 0
PCIBusNumber = 3
PCIDeviceNumber = 11
LocalBusLeft = "Slot4"
LocalBusRight = "Slot6"
ExternalBackplaneInterface = "None"

[Chassis2Slot6]
PCISlotPath = "50,60,F0"
PCISlotPathRootBus = 0
PCIBusNumber = 3
PCIDeviceNumber = 10
LocalBusLeft = "Slot5"
LocalBusRight = "Slot7"
ExternalBackplaneInterface = "None"
```

```
[Chassis2PCIBusSegment2]
SlotList = "7,8,9,10,11,12"

[Chassis2TriggerBus2]
SlotList = "7,8,9,10,11,12"

[Chassis2Slot7]
PCISlotPath = "78,60,60,F0"
PCISlotPathRootBus = 0
PCIBusNumber = 4
PCIDeviceNumber = 15
LocalBusLeft = "Slot6"
LocalBusRight = "Slot8"
ExternalBackplaneInterface = "None"

[Chassis2Slot8]
PCISlotPath = "70,60,60,F0"
PCISlotPathRootBus = 0
PCIBusNumber = 4
PCIDeviceNumber = 14
LocalBusLeft = "Slot7"
LocalBusRight = "Slot9"
ExternalBackplaneInterface = "None"

[Chassis2Slot9]
PCISlotPath = "68,60,60,F0"
PCISlotPathRootBus = 0
PCIBusNumber = 4
PCIDeviceNumber = 13
LocalBusLeft = "Slot8"
LocalBusRight = "Slot10"
ExternalBackplaneInterface = "None"

[Chassis2Slot10]
PCISlotPath = "58,60,60,F0"
PCISlotPathRootBus = 0
PCIBusNumber = 4
PCIDeviceNumber = 11
LocalBusLeft = "Slot9"
LocalBusRight = "Slot11"
ExternalBackplaneInterface = "None"

[Chassis2Slot11]
PCISlotPath = "50,60,60,F0"
PCISlotPathRootBus = 0
PCIBusNumber = 4
PCIDeviceNumber = 10
LocalBusLeft = "Slot10"
LocalBusRight = "Slot12"
ExternalBackplaneInterface = "None"

[Chassis2Slot12]
PCISlotPath = "48,60,60,F0"
PCISlotPathRootBus = 0
```

```
PCIBusNumber = 4
PCIDeviceNumber = 9
LocalBusLeft = "Slot11"
LocalBusRight = "Slot13"
ExternalBackplaneInterface = "None"

[Chassis2PCIBusSegment3]
SlotList = "13,14,15,16,17,18"

[Chassis2TriggerBus3]
SlotList = "13,14,15,16,17,18"

[Chassis2Slot13]
PCISlotPath = "78,60,60,60,F0"
PCISlotPathRootBus = 0
PCIBusNumber = 5
PCIDeviceNumber = 15
LocalBusLeft = "Slot12"
LocalBusRight = "Slot14"
ExternalBackplaneInterface = "None"

[Chassis2Slot14]
PCISlotPath = "70,60,60,60,F0"
PCISlotPathRootBus = 0
PCIBusNumber = 5
PCIDeviceNumber = 14
LocalBusLeft = "Slot13"
LocalBusRight = "Slot15"
ExternalBackplaneInterface = "None"

[Chassis2Slot15]
PCISlotPath = "68,60,60,60,F0"
PCISlotPathRootBus = 0
PCIBusNumber = 5
PCIDeviceNumber = 13
LocalBusLeft = "Slot14"
LocalBusRight = "Slot16"
ExternalBackplaneInterface = "None"

[Chassis2Slot16]
PCISlotPath = "60,60,60,60,F0"
PCISlotPathRootBus = 0
PCIBusNumber = 5
PCIDeviceNumber = 12
LocalBusLeft = "Slot15"
LocalBusRight = "Slot17"
ExternalBackplaneInterface = "None"

[Chassis2Slot17]
PCISlotPath = "58,60,60,60,F0"
PCISlotPathRootBus = 0
PCIBusNumber = 5
PCIDeviceNumber = 11
LocalBusLeft = "Slot16"
```

```
LocalBusRight = "Slot18"
ExternalBackplaneInterface = "None"

[Chassis2Slot18]
PCISlotPath = "50,60,60,60,F0"
PCISlotPathRootBus = 0
PCIBusNumber = 5
PCIDeviceNumber = 10
LocalBusLeft = "Slot17"
LocalBusRight = "None"
ExternalBackplaneInterface = "None"
```

2.4 Chassis Description Files

Chassis description files characterize PXI chassis. The primary purpose of a chassis description file is to enumerate PCI bus segments, trigger buses, trigger bridges, sets of star triggers, and slots. Chassis description files are a key component in the PXI hardware description architecture, enabling a Resource Manager to generate a PXI system description.

RULE: A chassis manufacturer SHALL provide a chassis description file for each chassis model produced.

RULE: A chassis description file SHALL be named `vendorDefinedText.ini`, where `vendorDefinedText` is a vendor-defined string used to uniquely name a chassis description file.

RULE: A chassis description file name SHALL contain the name of the chassis vendor to guarantee uniqueness versus chassis description files from other vendors.

RULE: To maximize backward compatibility, a Resource Manager SHALL be capable of reading chassis description files with any filename ending with *.ini*.

RULE: A system controller module SHALL provide the following Windows registry value in the 32-bit registry for specifying a location of chassis description files:

Key: HKEY_LOCAL_MACHINE\SOFTWARE\PXISA\CurrentVersion

Value: ChassisDescriptionFilePath

The ChassisDescriptionFilePath SHALL be a string value that specifies the complete path of a directory that holds chassis description files.

RULE: Installation software for a chassis description file SHALL NOT delete or modify the ChassisDescriptionFilePath registry key if it already exists.

RULE: When creating the ChassisDescriptionFilePath on a system where it did not previously exist, an installer SHALL set the value to the directory `%ALLUSERSAPPDATA%\PXISA\Descriptions\Chassis\`, where `%ALLUSERSAPPDATA%` is the user-independent application data folder.

OBSERVATION: Prior versions of this specification did not dictate a specific folder for the chassis description files, but allowed installers to install a registry key to point to an arbitrary directory. While the above rule was added to simplify installation and removal of software components, the registry key mechanism is maintained for backward compatibility.

OBSERVATION: Using the ChassisDescriptionFilePath registry value, chassis description file installers can copy their chassis description files to a standard location. In addition, a PXI Resource Manager can use this location to identify the types of chassis available for a PXI system.

PERMISSION: A vendor MAY place descriptors or tags in a chassis description file other than those described in this section.

OBSERVATION: The above permission may be useful to store supplemental information about a chassis that is useful for advanced vendor-specific functionality.

RECOMMENDATION: Any vendor-specific descriptors or tags in a chassis description file SHOULD be named such that they are unlikely to collide with tags or descriptors added in a future version of any PXISA specification.

2.4.1 Chassis Description Definitions

To develop a chassis description, it is useful to define descriptors for the following chassis components:

- **Chassis** – A chassis descriptor corresponds to a physical PXI chassis. Chassis can include PCI bus segments, trigger buses, trigger bridges, star triggers, and slots. Line mapping specifications may be used to identify chassis capabilities to software.
  - **PCI Bus Segments** – A PCI bus segment descriptor corresponds to a distinct, physical PCI bus in a chassis. PCI bus segments can contain slots, bridges, and other backplane devices. Multiple PCI bus segments are linked within a chassis using PCI-PCI bridging.
  - **Trigger Buses** – A PXI trigger bus descriptor corresponds to a physical trigger bus in a PXI chassis. A trigger bus is characterized by a list of slots sharing the physical trigger bus connection. Chassis can contain multiple trigger buses.
  - **Trigger Bridges** – A PXI trigger bridge descriptor corresponds to a physical trigger bridge in a PXI chassis. Each trigger bridge descriptor represents the possible unidirectional routes that can be established between two buses; if a physical trigger bridge can be used to establish routes in either direction between these buses, two trigger bridge descriptors must represent it, one for each direction. A chassis can contain multiple trigger bridges.
  - **Line Mapping Specifications** – A line mapping specification does not represent a physical chassis component, but sets out the possible routes that can be established by a trigger bridge between two adjacent trigger buses. This line mapping provides software with detailed information about the routing capabilities that the chassis supports. These routes can be established through calls made to the chassis Trigger Manager, as described in *PXI-9: PXI and PXI Express Trigger Management Specification*. Multiple line mappings can describe a chassis' routing capabilities.
  - **Star Triggers –** A PXI star trigger descriptor corresponds to a physical set of star triggers in a chassis. A set of star triggers is characterized by a star trigger controller slot number and a mapping of PXI_STAR lines to peripheral slot numbers. A chassis can contain multiple sets of star triggers.
  - **Bridges** – A bridge descriptor corresponds to a physical PCI-PCI bridge. A bridge is characterized by a secondary bus segment (that is, the PCI bus segment subordinate to the bridge). A chassis can have multiple bridges.
  - **Slots** –A PXI slot descriptor corresponds to a physical slot in a chassis. A slot is characterized by a geographic address, a PCI logical address, local bus routings, and other special capabilities. A chassis has multiple slots.

2.4.2 Chassis Descriptor

A chassis descriptor provides a high-level description of a PXI chassis. A chassis descriptor contains collections of the components that comprise a chassis, including PCI bus segments, trigger buses, sets of star triggers, and slots.

RULE: A chassis description file SHALL contain one and only one chassis descriptor.

RULE: The chassis descriptor section SHALL be named “Chassis”.

RULE: Each chassis descriptor section SHALL contain one of each of the nonshaded tag line types described in Table 2-11.

RULE: The chassis descriptor section SHALL contain one of each of the shaded tag line types described in Table 2-11 if the chassis supports trigger routing as described in *PXI-9: PXI and PXI Express Trigger Management Specification*.

Table 2-11. Chassis Description File – Chassis Tag Line Descriptions

Tag	Valid Values	Description
PCIBusSegmentList	A comma-separated list of n, where *n* is a decimal integer such that 1 <= *n* <= 255.	This tag enumerates the PCI bus segments in a chassis.
TriggerBusList	A comma-separated list of *n*, where *n* is a decimal integer such that *n* >= 1.	This tag enumerates the trigger buses in a chassis.
TriggerBridgeList	A comma-separated list of *n*, where *n* is a decimal integer such that *n* >= 1.	This tag enumerates the trigger bridges in a chassis.
LineMappingSpecList	A comma-separated list of *n*, where *n* is a decimal integer such that *n* >= 1.	This tag enumerates the line mapping specifications that exist in the chassis description file.
StarTriggerList	A comma-separated list of *n*, where *n* is a decimal integer such that *n* >= 1.	This tag enumerates the sets of star trigger in a chassis.
SlotList	A comma-separated list of *n*, where *n* is a decimal integer such that *n* >= 1.	This tag enumerates the slots in a chassis.
Model	A string indicating the model of this chassis.	This tag identifies the chassis model name.
Vendor	A string indicating the vendor of this chassis.	This tag identifies the chassis vendor name.

Chassis Descriptor Example

```
# This example describe a 3-segment, 18-slot PXI chassis
[Chassis]
PCIBusSegmentList = "1,2,3"
TriggerBusList = "1,2,3"
TriggerBridgeList = "1,2,3"
LineMappingSpecList = "1,2"
StarTriggerList = "1"
SlotList = "1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18"
Model = "18-Slot Chassis"
Vendor = "PXISA"
```

RULE: Multiple PCI bus segments SHALL be uniquely numbered in the PCIBusSegmentList tag.

OBSERVATION: PCI bus segments can be numbered in an arbitrary fashion. For example, PCI bus segments can be numbered according to their order of discovery using a depth-first PCI traversal algorithm.

RULE: Multiple trigger buses SHALL be uniquely numbered in the TriggerBusList tag.

OBSERVATION: Trigger buses can be numbered in an arbitrary fashion. For example, a trigger bus can be sequentially numbered based on the relative order of the slots it contains.

RULE: Multiple trigger bridges SHALL be uniquely numbered in the TriggerBridgeList tag.

OBSERVATION: Trigger bridges can be numbered in an arbitrary fashion.

RULE: Multiple line mapping specifications SHALL be uniquely numbered in the LineMappingSpecList tag.

OBSERVATION: Line mapping specifications can be numbered in an arbitrary fashion.

RULE: Multiple sets of star triggers SHALL be uniquely numbered in the StarTriggerList tag.

OBSERVATION: Sets of star triggers can be numbered in an arbitrary fashion. For example, a set of star triggers can be sequentially numbered based on the relative order of the slots the set contains.

RULE: PXI slots SHALL be uniquely numbered according to their corresponding physically-viewable slot numbers.

2.4.3 PCI Bus Segment Descriptor

A PCI bus segment descriptor characterizes a PCI bus segment in a chassis. The most important aspect of a PCI bus segment descriptor is that it describes the mapping from PCI address lines (AD[31:0]) to IDSEL assignments for the segment's slots, bridges, and backplane devices.

RULE: A chassis description file SHALL contain a distinct PCI bus segment descriptor for each physical PCI bus segment in a chassis.

RULE: A PCI bus segment descriptor SHALL be named "PCIBusSegment*N*", where *N* is the PCI bus segment number.

RULE: PCI bus segment numbers SHALL be derived from the PCIBusSegmentBusList tag of the chassis descriptor (see Table 2-11).

OBSERVATION: While each PCI bus segment number will uniquely correspond to a PCI bus number, the PCI bus segment number will not necessarily be equal to the corresponding PCI bus number.

RULE: Each PCI bus segment descriptor SHALL contain one of each of the tag line types described in Table 2-12.

Table 2-12. Chassis Description File – PCI Bus Segment Tag Line Descriptions

Tag	Valid Values	Description
SlotList	A comma-separated list of *n*, where *n* is a decimal integer such that *n* >= 1.	This tag enumerates the slots on a PCI bus segment.
BridgeList	A comma-separated list of *n*, where *n* is a decimal integer such that 1 <= *n* <= 255.	This tag enumerates the PCI-PCI bridges on a PCI bus segment.

Table 2-12. Chassis Description File – PCI Bus Segment Tag Line Descriptions (Continued)

Tag	Valid Values	Description
IDSELList	*n*, where *n* is a decimal integer such that 1 <= *n* <= 31.	This tag lists the PCI address line numbers (AD[31:0]) used to implement the IDSEL signals for devices on a PCI bus segment.
IDSEL*n*, where *n* is a decimal integer corresponding to a PCI address line (AD[31:0]), for each *n* contained in the IDSELList	A slot descriptor. A bridge descriptor. (Other).	This tag specifies the PCI address line number (AD[31:0]) used to implement the IDSEL signal for a given slot, bridge, or backplane device on a PCI bus segment.

PCI Bus Segment Descriptor Example

```
# This example describes the first PCI bus segment of a
# 3-segment, 18-slot chassis.
[PCIBusSegment1]
SlotList = "1,2,3,4,5,6"
BridgeList = "1"
IDSELList = "31,30,29,28,27,26"
IDSEL31 = "Slot2"
IDSEL30 = "Slot3"
IDSEL29 = "Slot4"
IDSEL28 = "Bridge1"
IDSEL27 = "Slot5"
IDSEL26 = "Slot6"
```

RULE: Slots SHALL be uniquely numbered in the SlotList tag.

OBSERVATION: Slot numbers will correspond to physically-viewable slot numbers for a PCI bus segment. In addition, the SlotList will be a subset of the SlotList specified in the chassis descriptor (see Table 2-11).

RULE: Multiple bridges SHALL be uniquely numbered in the BridgeList tag.

OBSERVATION: Bridges can be numbered in an arbitrary fashion. For example, bridges can be numbered according to their order of discovery using a depth-first PCI traversal algorithm.

PERMISSION: A PCI bus segment descriptor MAY contain an empty list of slots. For example, a chassis might use multiple levels of PCI-PCI bridging to bridge two PXI bus segments. In this case, the first of these PCI bus segments would not contain a list of slots. This type of PCI bus segment will contain a bridge routing, however.

PERMISSION: A PCI bus segment descriptor MAY specify an IDSEL routing to a backplane device other than a slot or a bridge.

2.4.3.1 System Controller Module Slot Considerations

The system controller module slot presents a special challenge in describing PCI bus segments.

OBSERVATION: Because the system controller module does not have its own IDSEL assignment, it is not included in the IDSELList. The slot number of the system controller module is included in the SlotList tag for a chassis' first PCI bus segment descriptor, however, and a Resource Manager can enumerate the system controller module in this manner.

2.4.4 Trigger Bus Descriptor

A trigger bus descriptor describes an individual trigger bus in a PXI chassis. A trigger bus is characterized by a list of slots that reside on the trigger bus.

RULE: A chassis description file SHALL contain a distinct PXI trigger bus descriptor for each physical PXI trigger bus in the chassis.

RULE: A trigger bus descriptor SHALL be named "TriggerBus*N*", where *N* is the trigger bus number.

RULE: Trigger bus numbers SHALL be derived from the TriggerBusList tag of the chassis descriptor (see Table 2-11).

OBSERVATION: While each trigger bus number will uniquely correspond to a set of PXI slots, there is not necessarily a one-to-one correspondence between trigger buses and PCI bus segments.

RULE: Each trigger bus descriptor SHALL contain one of each of the tag line types described in Table 2-13.

Table 2-13. Chassis Description File – Trigger Bus Tag Line Descriptions

Tag	Valid Values	Description
SlotList	A comma-separated list of *n*, where *n* is a decimal integer such that *n* >= 1.	This tag enumerates the slots on a trigger bus.

Trigger Bus Descriptor Example

```
# This example describes the first trigger bus segment of a
# 3-segment, 18-slot chassis.
[TriggerBus1]
SlotList = "1,2,3,4,5,6"
```

2.4.5 Trigger Bridge Descriptor

A trigger bridge descriptor describes a unidirectional trigger bridge in a PXI chassis.

RULE: A trigger bridge descriptor SHALL be named *TriggerBridgeN*, where *N* is the number of the trigger bridge.

RULE: Trigger Bridge descriptor numbers SHALL be derived from the TriggerBridgeList tag of the chassis descriptor in Table 2-11.

RULE: Each trigger bridge descriptor SHALL contain one of each of the tagline types described in Table 2-14.

Table 2-14. Chassis Description File – Trigger Bridge Descriptions

Tag	Valid Values	Description
SourceTriggerBus	*n*, where *n* is a decimal integer such that *n* >= 1.	The source trigger bus for this trigger bridge.

Table 2-14. Chassis Description File – Trigger Bridge Descriptions (Continued)

Tag	Valid Values	Description
DestinationTriggerBus	*n*, where *n* is a decimal integer such that *n* >= 1.	The destination trigger bus for this trigger bridge.
LineMappingSpec	*n*, where *n* is a decimal integer such that *n* >= 1.	The number of the line mapping spec that describes the routing capabilities of this trigger bridge.

Trigger Bridge Descriptor Example

```
# This example describes a trigger bridge that
# can route signals from trigger bus 1 to trigger
# bus 2, with line-by-line capabilities described by
# LineMappingSpec 1.
[TriggerBridge1]
SourceTriggerBus = 1
DestinationTriggerBus = 2
LineMappingSpec = 1
```

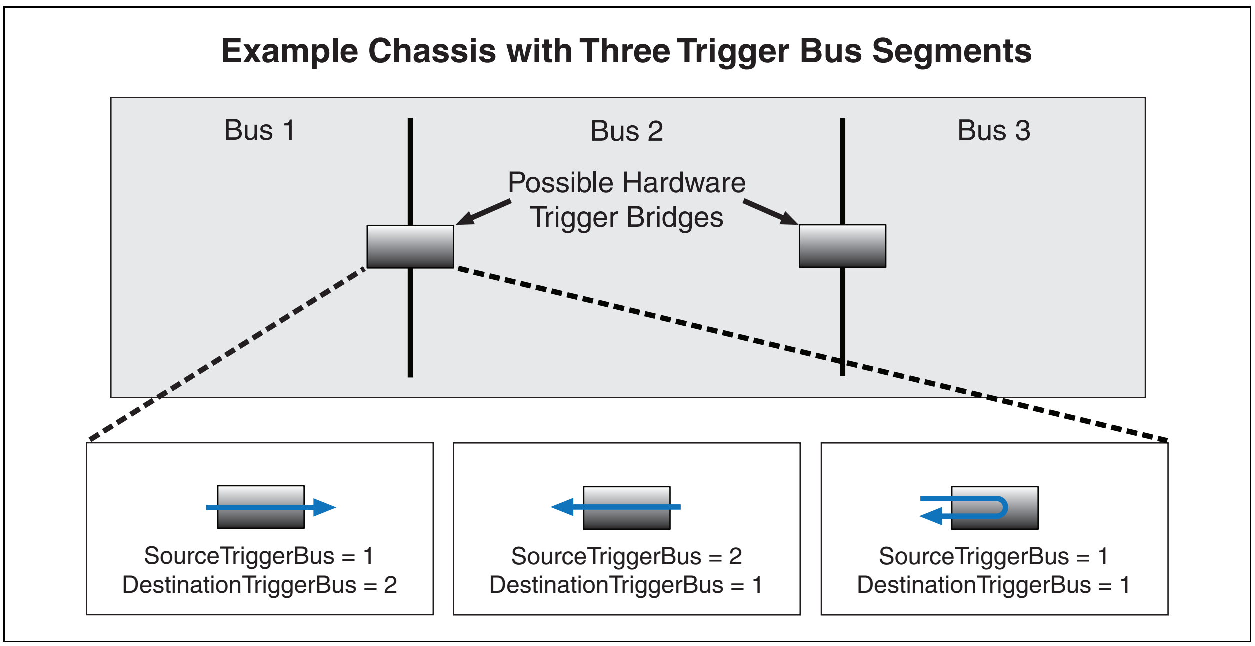

Figure 2-1. A hardware trigger bridge may have multiple software representations in the chassis description file, supporting one or more of the capabilities shown in this figure. It must have a separate trigger bridge descriptor for each software representation.

OBSERVATION: Because the trigger bridge descriptor has tags to describe unidirectional routing capabilities only, a hardware trigger bridge that can route triggers in both directions between two buses must be represented by two trigger bridge descriptors in the chassis description file. Refer to Figure 2-1 for an example.

OBSERVATION: It is possible to describe a trigger bridge where the source trigger bus and destination trigger bus are the same. In this case, the distinction between unidirectional and bidirectional routing capabilities is irrelevant. Refer to Figure 2-1 for an example.

RULE: The SourceTriggerBus and DestinationTriggerBus tag values SHALL describe two buses which are physically connected in the chassis by a single hardware trigger bridge, or the same bus where the lines of that bus are physically connected to each other by a single hardware trigger bridge.

OBSERVATION: The trigger bridge descriptor is intended to show direct routing capabilities that do not involve intermediate buses. To make routes involving intermediate buses, clients must ensure that those buses are not in use by other software in the system to ensure successful operation. Refer to *PXI-9: PXI and PXI Express Trigger Management Specification* for further information.

OBSERVATION: Trigger bridge descriptors are required only for chassis that contain hardware trigger bridges that support a programmable routing feature.

2.4.6 Line Mapping Spec Descriptor

A line mapping specification describes the possible routes that can be established between a given source bus and destination bus. The line mapping specification is in a separate descriptor so that it can be referenced from multiple trigger bridge descriptors, avoiding unnecessary duplication of information about routing capabilities.

OBSERVATION: There is no direct relationship between the number of physical trigger bridges in a chassis and the number of line mapping specification descriptors necessary; there should be as many line mapping spec descriptors as there are unique sets of bus-to-bus routing capabilities provided by trigger routers. For example, if a chassis has three trigger bridges with equivalent routing capabilities in each direction, then only a single line mapping spec descriptor would be necessary.

RULE: A line mapping specification descriptor SHALL be named *LineMappingSpecN*, where *N* is the number for the line mapping specification.

RULE: Line mapping spec descriptor numbers SHALL be derived from the LineMappingSpecList tag of the chassis descriptor (refer to Table 2-8).

RULE: Each line mapping spec descriptor SHALL contain one of each of the tagline types described in Table 2-15

Table 2-15. Chassis Description File – Line Mapping Spec Descriptions

Tag	Valid Values	Description
PXI_TRIG*n*, where *n* is an integer that represents a PXI trigger line on the source trigger bus of the trigger bridge referencing this descriptor. One tag must exist for each trigger line on the source bus.	A comma-separated list of *n*, where *n* is a decimal integer such that 0 <= *n* <= 7.	This tag enumerates the lines on the destination trigger bus to which the referencing trigger bridge can route a signal from the trigger line on the source bus indicated by the tag name.

Line Mapping Spec Descriptor Example

```
# This example describes a line mapping in which
# the referencing trigger bridge can map any line on the
# source trigger bus to any line on the destination
# trigger bus.
[LineMappingSpec1]
PXI_TRIG0 = "0,1,2,3,4,5,6,7"
PXI_TRIG1 = "0,1,2,3,4,5,6,7"
```

```
PXI_TRIG2 = "0,1,2,3,4,5,6,7"
PXI_TRIG3 = "0,1,2,3,4,5,6,7"
PXI_TRIG4 = "0,1,2,3,4,5,6,7"
PXI_TRIG5 = "0,1,2,3,4,5,6,7"
PXI_TRIG6 = "0,1,2,3,4,5,6,7"
PXI_TRIG7 = "0,1,2,3,4,5,6,7"
```

OBSERVATION: The presence of a line mapping from line X to line Y does not guarantee that a trigger route from line X on the source bus to line Y on the destination bus can be made at any given time. Even if line Y is an available destination, limited hardware resources in the trigger bridge component may prevent certain routes from taking place simultaneously. The line mapping spec descriptor is intended primarily as a hint to limit the search space for software that determines available routes at runtime.

OBSERVATION: Line mapping specification descriptors are required only for chassis that contain hardware trigger bridges that support a programmable routing feature, which is not a requirement.

2.4.7 Star Trigger Descriptor

A star trigger descriptor describes an individual set of star triggers in a PXI chassis. A star trigger descriptor is characterized by a star trigger controller slot number and a mapping of PXI_STAR lines, as defined in the *PXI Hardware Specification*, to peripheral slot numbers.

RULE: A chassis description file SHALL contain a distinct PXI star trigger descriptor for each physical set of star triggers in the chassis.

RULE: A star trigger descriptor SHALL be named "StarTrigger*N*", where *N* is the number for the set of star triggers.

RULE: Star trigger descriptor numbers SHALL be derived from the StarTriggerList tag of the chassis descriptor (see Table 2-11).

RULE: Each star trigger descriptor SHALL contain one of each of the tag line types described in Table 2-16.

Table 2-16. Chassis Description File – Star Trigger Tag Line Descriptions

Tag	Valid Values	Description
ControllerSlot	A decimal integer *n*, where *n* is a decimal integer such that *n* >= 1.	This tag specifies the star trigger controller slot number for a set of star triggers.
PXI_STAR*n* (where *n* is a decimal integer such that 0 <= *n* <= 12), for each PXI star trigger line routed to a PXI slot.	A decimal integer *m*, where *m* is a valid PXI slot number that connects to the star trigger line.	This tag specifies the PXI_STAR line to slot number mapping for a set of star triggers.

Star Trigger Descriptor Example

```
# This example describes the star trigger bus of a
# 3-segment, 18-slot PXI chassis.
[StarTrigger1]
ControllerSlot = 2
PXI_STAR0 = 3
PXI_STAR1 = 4
PXI_STAR2 = 5
PXI_STAR3 = 6
PXI_STAR4 = 7
```

```
PXI_STAR5 = 8
PXI_STAR6 = 9
PXI_STAR7 = 10
PXI_STAR8 = 11
PXI_STAR9 = 12
PXI_STAR10 = 13
PXI_STAR11 = 14
PXI_STAR12 = 15
```

2.4.8 Bridge Descriptor

A bridge descriptor characterizes PCI-PCI bridges linking two PCI bus segments in a multisegment PXI chassis. A bridge contains a pointer to a secondary bus segment (the PCI bus segment subordinate to the bridge).

RULE: A chassis description file SHALL contain a distinct bridge descriptor for each PCI-PCI bridge linking multiple PCI bus segments.

RULE: A bridge descriptor SHALL be named "Bridge*N*", where *N* is the bridge number.

RULE: Bridge numbers SHALL be derived from the BridgeList tag of the PCI bus segment descriptor (see Table 2-12).

RULE: Each bridge descriptor SHALL contain one of each of the tag line types described in Table 2-17.

Table 2-17. Chassis Description File – Bridge Tag Line Descriptions

Tag	Valid Values	Description
SecondaryBusSegment	A PCI bus segment descriptor for the segment subordinate to this bridge.	This tag points to the PCI bus segment subordinate to this bridge.

Bridge Descriptor Example

```
# This example describes a bridge that links segment 1
# to segment 2 in a 3-segment PXI chassis.
[Bridge1]
SecondaryBusSegment = "PCIBusSegment2"
```

2.4.9 Slot Descriptor

A slot descriptor describes an individual slot in a PXI chassis. A slot descriptor is characterized by the features of the slot it describes, including routing information for the slot's local bus lines.

RULE: A chassis description file SHALL contain a distinct slot descriptor for each physical slot in the chassis.

RULE: A slot descriptor SHALL be named "Slot*N*", where *N* is the physical slot number.

RULE: A slot number SHALL be derived by enumerating IDSEL assignments for the corresponding PCI bus segment descriptor (see Table 2-12).

PERMISSION: A slot number MAY be derived from alternate sources, including a chassis descriptor's SlotList tag (see Table 2-11) or the corresponding PCI bus segment descriptor's SlotList tag (see Table 2-12).

RULE: Each slot descriptor SHALL contain one of each of the non-shaded tag line types described in Table 2-18.

PERMISSION: A chassis description MAY continue to use the shaded fields of Table 2-18.These fields may be removed in a future revision.

Table 2-18. Chassis Description File – Slot Tag Line Descriptions

Tag	Valid Values	Description
LocalBusLeft	A valid slot descriptor. A valid star trigger descriptor. (Other).	This tag indicates how this slot routes its local bus pins to the left.
LocalBusRight	A valid slot descriptor. (Other).	This tag indicates how this slot routes its local bus pins to the right.
ExternalBackplaneInterface	None. (Other).	If this slot routes to an external backplane interface, this tag specifies the name of that interface.

Slot Descriptor Example

```
# This example describes a PXI slot in a 3-segment, 18-slot
# PXI chassis.
[Slot12]
LocalBusLeft = "Slot11"
LocalBusRight = "Slot13"
ExternalBackplaneInterface = "None"
```

OBSERVATION: A PXI slot that does not implement the full set of PXI features, such as a CompactPCI-only slot, will have tag values corresponding to PXI features set to “None”. For example, a CompactPCI-only slot would set its LocalBusLeft and LocalBusRight tags to “None”. In addition, this slot would not be present in the SlotList for a trigger bus, and it would not belong to a set of star triggers.

2.4.10 Chassis Description File Examples

The following are complete examples of chassis description files.

2.4.10.1 Example 8-Slot PXI Chassis

```
# This example describes an 8-slot PXI chassis with
# 1 PCI bus segment, 1 trigger bus, and 1 star trigger.

[Version]
Major = 2
Minor = 4

[Chassis]
Model = "Example 8-Slot Chassis"
Vendor = "PXISA"
PCIBusSegmentList = "1"
TriggerBusList = "1"
```

```
StarTriggerList = "1"
SlotList = "1,2,3,4,5,6,7,8"

[PCIBusSegment1]
SlotList = "1,2,3,4,5,6,7,8"
BridgeList = "None"
IDSELList = "31,30,29,28,27,26,25"
IDSEL31 = "Slot2"
IDSEL30 = "Slot3"
IDSEL29 = "Slot4"
IDSEL28 = "Slot5"
IDSEL27 = "Slot6"
IDSEL26 = "Slot7"
IDSEL25 = "Slot8"

[TriggerBus1]
SlotList = "1,2,3,4,5,6,7,8"

[StarTrigger1]
ControllerSlot = 2
PXI_STAR0 = 3
PXI_STAR1 = 4
PXI_STAR2 = 5
PXI_STAR3 = 6
PXI_STAR4 = 7
PXI_STAR5 = 8

[Slot1]
LocalBusLeft = "None"
LocalBusRight = "None"
ExternalBackplaneInterface = "None"

[Slot2]
LocalBusLeft = "StarTrigger1"
LocalBusRight = "Slot3"
ExternalBackplaneInterface = "None"

[Slot3]
LocalBusLeft = "Slot2"
LocalBusRight = "Slot4"
ExternalBackplaneInterface = "None"

[Slot4]
LocalBusLeft = "Slot3"
LocalBusRight = "Slot5"
ExternalBackplaneInterface = "None"

[Slot5]
LocalBusLeft = "Slot4"
LocalBusRight = "Slot6"
ExternalBackplaneInterface = "None"

[Slot6]
LocalBusLeft = "Slot5"
```

```
LocalBusRight = "Slot7"
ExternalBackplaneInterface = "None"

[Slot7]
LocalBusLeft = "Slot6"
LocalBusRight = "Slot8"
ExternalBackplaneInterface = "None"

[Slot8]
LocalBusLeft = "Slot7"
LocalBusRight = "None"
ExternalBackplaneInterface = "None"
```

2.4.10.2 Example 18-Slot PXI Chassis

```
# This example describes an 18-slot PXI chassis with
# 3 PCI bus segment, 3 trigger buses, and 1 star
# trigger. The chassis has a bidirectional trigger
# bridge between trigger buses 1 and 2 which can route
# any line on one bus to any line on the other bus.
# Additionally, the chassis has a unidirectional
# trigger bridge that can route any line on bus 2
# to the same line on bus 3.

[Version]
Major = 2
Minor = 4

[Chassis]
Model = "Example 18-Slot Chassis"
Vendor = "PXISA"
PCIBusSegmentList = "1,2,3"
TriggerBusList = "1,2,3"
TriggerBridgeList = "1,2,3"
LineMappingSpec = "1,2"
StarTriggerList = "1"
SlotList = "1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18"

[PCIBusSegment1]
SlotList = "1,2,3,4,5,6"
BridgeList = "1"
IDSELList = "31,30,29,28,27,26"
IDSEL31 = "Slot2"
IDSEL30 = "Slot3"
IDSEL29 = "Slot4"
IDSEL28 = "Bridge1"
IDSEL27 = "Slot5"
IDSEL26 = "Slot6"

[TriggerBus1]
SlotList = "1,2,3,4,5,6"

[StarTrigger1]
ControllerSlot = 2
```

```
PXI_STAR0 = 3
PXI_STAR1 = 4
PXI_STAR2 = 5
PXI_STAR3 = 6
PXI_STAR4 = 7
PXI_STAR5 = 8
PXI_STAR6 = 9
PXI_STAR7 = 10
PXI_STAR8 = 11
PXI_STAR9 = 12
PXI_STAR10 = 13
PXI_STAR11 = 14
PXI_STAR12 = 15

[Slot1]
LocalBusLeft = "None"
LocalBusRight = "None"
ExternalBackplaneInterface = "None"

[Slot2]
LocalBusLeft = "StarTrigger1"
LocalBusRight = "Slot3"
ExternalBackplaneInterface = "None"

[Slot3]
LocalBusLeft = "Slot2"
LocalBusRight = "Slot4"
ExternalBackplaneInterface = "None"

[Slot4]
LocalBusLeft = "Slot3"
LocalBusRight = "Slot5"
ExternalBackplaneInterface = "None"

[Slot5]
LocalBusLeft = "Slot4"
LocalBusRight = "Slot6"
ExternalBackplaneInterface = "None"

[Slot6]
LocalBusLeft = "Slot5"
LocalBusRight = "Slot7"
ExternalBackplaneInterface = "None"

[Bridge1]
SecondaryBusSegment = "PCIBusSegment2"

[PCIBusSegment2]
SlotList = "7,8,9,10,11,12"
BridgeList = "2"
IDSELList = "31,30,29,28,27,26,25"
IDSEL31 = "Slot7"
IDSEL30 = "Slot8"
IDSEL29 = "Slot9"
```

```
IDSEL28 = "Bridge2"
IDSEL27 = "Slot10"
IDSEL26 = "Slot11"
IDSEL25 = "Slot12"

[TriggerBus2]
SlotList = "7,8,9,10,11,12"

[Slot7]
LocalBusLeft = "Slot6"
LocalBusRight = "Slot8"
ExternalBackplaneInterface = "None"

[Slot8]
LocalBusLeft = "Slot7"
LocalBusRight = "Slot9"
ExternalBackplaneInterface = "None"

[Slot9]
LocalBusLeft = "Slot8"
LocalBusRight = "Slot10"
ExternalBackplaneInterface = "None"

[Slot10]
LocalBusLeft = "Slot9"
LocalBusRight = "Slot11"
ExternalBackplaneInterface = "None"

[Slot11]
LocalBusLeft = "Slot10"
LocalBusRight = "Slot12"
ExternalBackplaneInterface = "None"

[Slot12]
LocalBusLeft = "Slot11"
LocalBusRight = "Slot13"
ExternalBackplaneInterface = "None"

[Bridge2]
SecondaryBusSegment = "PCIBusSegment3"

[PCIBusSegment3]
SlotList = "13,14,15,16,17,18"
BridgeList = "None"
IDSELList = "31,30,29,28,27,26"
IDSEL31 = "Slot13"
IDSEL30 = "Slot14"
IDSEL29 = "Slot15"
IDSEL28 = "Slot16"
IDSEL27 = "Slot17"
IDSEL26 = "Slot18"

[TriggerBus3]
SlotList = "13,14,15,16,17,18"
```

```
[Slot13]
LocalBusLeft = "Slot12"
LocalBusRight = "Slot14"
ExternalBackplaneInterface = "None"

[Slot14]
LocalBusLeft = "Slot13"
LocalBusRight = "Slot15"
ExternalBackplaneInterface = "None"

[Slot15]
LocalBusLeft = "Slot14"
LocalBusRight = "Slot16"
ExternalBackplaneInterface = "None"

[Slot16]
LocalBusLeft = "Slot15"
LocalBusRight = "Slot17
ExternalBackplaneInterface = "None"

[Slot17]
LocalBusLeft = "Slot16"
LocalBusRight = "Slot18"
ExternalBackplaneInterface = "None"

[Slot18]
LocalBusLeft = "Slot17"
LocalBusRight = "None"
ExternalBackplaneInterface = "None"

[TriggerBridge1]
SourceTriggerBus = 1
DestinationTriggerBus = 2
LineMappingSpec = 1

[TriggerBridge2]
SourceTriggerBus = 2
DestinationTriggerBus = 1
LineMappingSpec = 1

[TriggerBridge3]
SourceTriggerBus = 2
DestinationTriggerBus = 3
LineMappingSpec = 2

[LineMappingSpec1]
PXI_TRIG0 = "0,1,2,3,4,5,6,7"
PXI_TRIG1 = "0,1,2,3,4,5,6,7"
PXI_TRIG2 = "0,1,2,3,4,5,6,7"
PXI_TRIG3 = "0,1,2,3,4,5,6,7"
PXI_TRIG4 = "0,1,2,3,4,5,6,7"
PXI_TRIG5 = "0,1,2,3,4,5,6,7"
PXI_TRIG6 = "0,1,2,3,4,5,6,7"
```

```
PXI_TRIG7 = "0,1,2,3,4,5,6,7"

[LineMappingSpec2]
PXI_TRIG0 = "0"
PXI_TRIG1 = "1"
PXI_TRIG2 = "2"
PXI_TRIG3 = "3"
PXI_TRIG4 = "4"
PXI_TRIG5 = "5"
PXI_TRIG6 = "6"
PXI_TRIG7 = "7"
```

This Page Intentionally Left Blank

3. Software Frameworks and Requirements

This section discusses the software features associated with a PXI system. It gives an overview of the general motivating factors behind the *PXI Software Specification*, along with specific software frameworks.

3.1 Overview

Like other bus architectures, PXI defines standards that allow products from multiple vendors to work together at the bus level. The *PXI Software Specification* goes on to mandate software requirements in addition to these bus level requirements. Other buses that have failed to designate software standards have seen the market fragment into competing standards from multiple vendors.

3.2 Motivation

Low-cost, rugged, reliable computer systems are needed in instrumentation and automation applications. The demands of reducing product cost and time to market, while increasing reliability, are severely straining custom-built systems. Hardware vendors have implemented many modular, multivendor solutions to tackle these problems. However, the majority of costs associated with any system development are more likely related to the software development time, rather than the hardware development time.

PXI, unlike other standards, defines *software frameworks* as part of its specification. These software frameworks ensure that a user has a complete, multivendor system solution from the start.

The frameworks that have been chosen for this specification reflect the dominance of these operating systems on current desktop PCs. As other operating systems become widely accepted and offer the same degree of software leverage as the current frameworks, they may be added to the supported PXI frameworks. Each framework is required to support the VISA software standard. VISA provides an industry-standard mechanism for locating and controlling PXI modules.

The currently supported frameworks are the 32-bit Windows and 64-bit Windows frameworks. Many organizations have aligned their entire offices around Microsoft operating systems, because of the reduction in training and support costs that come from a common, familiar computing environment across the office. These same benefits extend to the factory floor or test department.

Thousands of 32-bit and 64-bit applications are running today on the Windows platforms, ranging from technical and engineering applications to complete manufacturing and financial solutions. All of these tools can be leveraged to improve instrumentation systems.

The ever-increasing interconnection between computers that design, produce, and test goods is driving the requirement that these systems easily interact with each other. The use of a common operating system across these computers means that a richer set of tools is available for exchanging and sharing data. All of the work being done to interconnect the desktop—COM, ODBC, .NET, and so on—is now available to interconnect machines on the factory floor to each other and to the rest of the corporation.

3.3 Framework Definition

The software frameworks define PXI system software requirements for both system controller modules and PXI peripheral modules. System controller modules and PXI peripheral modules have to meet certain requirements for operating system and tool support in order to be considered compliant with a given PXI software framework.

RULE: PXI system controller modules and PXI peripheral modules SHALL support the 32-bit Windows framework or 64-bit Windows framework .

3.4 32-bit Windows System Framework

3.4.1 Introduction

This section defines the specific requirements for the 32-bit Windows system framework. It defines all of the unique components that must exist to support this framework. It also describes the optional recommended components.

3.4.2 Overview of the Framework

The 32-bit Windows system framework defines a system based on the popular PC architecture, and is based on the Windows operating system from Microsoft.

3.4.3 Controller Requirements

This section defines the system requirements for the 32-bit Windows framework for the system controller module.

RULE: The system controller module SHALL be based on the 80x86 architecture.

OBSERVATION: Processors other than the 80x86 that are supported under 32-bit Windows may be added in additional frameworks.

RULE: All system controller modules SHALL be supplied with a VISA implementation that supports the PXI bus and is consistent with the IVI Foundation VISA Library specification (VPP-4.3).

3.4.4 PXI Peripheral Module Requirements

Hardware vendors for other industrial buses that do not have software standards often do not provide any software drivers for their modules. The customer is often given only a manual describing how to write software to control the module. The cost to the customer, in terms of engineering effort to support these modules, is huge. PXI removes this burden by requiring that manufacturers, rather than customers, develop this software.

RULE: Peripheral modules SHALL provide software for installing, configuring, and controlling the modules under Windows.

RECOMMENDATION: PXI peripheral modules that are instrumentation class modules SHOULD provide a user-level interface that is supported under the development environments specified in Table 3-1.

Table 3-1. Development Environments Supported by PXI Modules Under Windows

Product	Company	
LabVIEW	National Instruments	
LabWindows/CVI	National Instruments	
ATEasy	Geotest-Marvin Test Systems, Inc.	
Visual Basic	Microsoft	
Visual C/C++	Microsoft	

PERMISSION: Other system tools MAY be supported in addition to these tools.

3.5 64-bit Windows System Framework

3.5.1 Introduction

This section defines the specific requirements for the 64-bit Windows system framework. It defines all the unique components that must exist to support this framework. It also describes the optional recommended components.

3.5.2 Overview of the Framework

The 64-bit Windows system framework defines a system based on the popular PC architecture, and is based on the Windows (x64) operating system from Microsoft.

3.5.3 Controller Requirements

This section defines the system requirements for the 64-bit Windows framework for the system controller module.

RULE: The system controller module SHALL be based on a 64-bit capable x86 architecture.

OBSERVATION: Processors other than 64-bit capable x86 that are supported under 64-bit Windows may be added in additional frameworks.

RULE: All system controller modules SHALL be supplied with a VISA implementation that supports the PXI bus and is consistent with the IVI Foundation VISA Library specification (VPP-4.3) version 4.0 or higher.

3.5.4 PXI Peripheral Module Requirements

Hardware vendors for other industrial buses that do not have software standards often do not provide any software drivers for their modules. The customer is often given only a manual describing how to write software to control the module. The cost to the customer, in terms of engineering effort to support these modules, is huge. PXI removes this burden by requiring that manufacturers, rather than customers, develop this software.

RULE: Peripheral modules SHALL provide software for installing, configuring, and controlling the modules under Windows.

RECOMMENDATION: PXI peripheral modules that are instrumentation class modules SHOULD provide a user-level interface that is supported under common development environments that support the 64-bit Windows system framework.

3.6 Support for Existing Instrumentation Standards

The challenge for developing PXI instrumentation systems is to provide a platform that extends the capabilities of new test systems, while supporting existing instrumentation standards. The IVI Foundation provides an industry-standard mechanism for communicating with GPIB, VXI, and serial instrumentation through a series of specifications. These specifications define communications standards (*VISA*), instrument programming interfaces (*instrument drivers*), and interactive user interfaces (*soft front panels*). These specifications also define frameworks for the various Windows operating systems.

RULE: A PXI module that controls a VISA supported interface other than the PXI bus SHALL provide the VISA software as a mechanism for communicating with that interface.

The use of the VISA standard in PXI preserves the user's investment in existing instrumentation software. VISA provides the link from PXI to a VXI chassis and instruments and standalone GPIB and serial instruments.

RECOMMENDATION: Instrumentation class PXI peripheral modules SHOULD provide instrument drivers and soft front panels that are consistent with the VXI*plug&play* instrument driver specifications (VPP-3.*x* and VPP-7).

The goal of supporting VXI*plug&play* instrument drivers and soft front panels is to provide a familiar development environment to test and measurement customers. Test and measurement customers have come to expect VXI and GPIB instruments to have soft front panels and instrument drivers. VXI*plug&play* support for native PXI instruments provides a seamless software path between VXI and PXI based systems.

RULE: All system controller modules SHALL be supplied with a VISA implementation that supports the PXI bus and is consistent with the IVI Foundation VISA Library specification (VPP-4.3) version 4.0 or higher.

4. Service Registration and Configuration

This section describes registration for a PXI Resource Manager and the configuration file used to select the PXI Resource Manager and default PXI Trigger Manager for a system.

4.1 Overview

As described in previous sections, it is recommended that a system controller vendor provide a software resource manager to generate the system description file. Because there may be multiple system controllers in a single PXI system, there can also be multiple resource managers installed at the same time. To avoid contention between resource managers over the contents of the system description file, a mechanism is necessary to identify a single resource manager as the owner of the file. This section describes the system configuration file, which is used to identify the active resource manager.

It is important to note that Resource Manager implementations can function such that changing the active Resource Manager from one vendor to another may require a user to re-enter some configuration information to maintain the previous configuration, such as user-selected chassis numbers or manual chassis identifications. This can be avoided if the newly activated Resource Manager uses the existing system description file as input before running its Resource Manager algorithm for the first time. This optimization is considered an implementation detail, and is not a requirement.

For an active resource manager to be selected, it is necessary to have a list of resource managers available on the system. This section describes a mechanism for registration of resource managers so that such a list is created.

As described in *PXI-9: PXI and PXI Express Trigger Management Specification*, some chassis have a trigger manager called out in the Services Tree either for the specific chassis model or more broadly for chassis created by a specific vendor. In cases where no such designation is made, such as chassis predating PXI-9, a default trigger manager is necessary to allow for the reservation of trigger lines in the chassis. This chapter also describes the selection of a default trigger manager, as recorded in the system configuration file.

4.2 Resource Manager Registration

Because multiple software resource managers may be installed on the same system, a mechanism is necessary to describe which Resource Managers are available on the system.

RULE: A software Resource Manager SHALL be registered in the Services Tree, as described in *PXI-9: PXI and PXI Express Trigger Management Specification* and *PXI-6: PXI Express Software Specification*.

RULE: A category key called “Resource Managers” SHALL be used to allow registration of software Resource Managers.

RULE: The installation software for a Resource Manager SHALL install a unique name key for the Resource Manager in the “Resource Managers” category key.

RULE: The string *None* SHALL NOT be used for the name of a Resource Manager.

RULE: The name of a Resource Manager SHALL include the name of the vendor of that Resource Manager.

OBSERVATION: The use of a name key allows a vendor to implement more than one Resource Manager.

RECOMMENDATION: A Resource Manager name SHOULD be user friendly, containing at least the vendor name, as well as some additional descriptive text if the vendor has provided multiple Resource Managers.

RULE: The name key for a Resource Manager SHALL contain attributes, whose type is Integer, and whose names are *PXI-nVersion*, where *n* is the specification number for each PXI specification that describes a behavior of the Resource Manager.

RULE: The value of each PXI-nVersion key SHALL be the version of the relevant specification to which the Resource Manager complies, expressed as a 32-bit number with the major version expressed in the top 16 bits, and the minor version expressed in the lower 16 bits.

As an example of a PXI-nVersion attribute, the value of version 1.2 would be expressed as 0x00010002.

OBSERVATION: Behaviors of a Resource Manager are described in PXI-2, PXI-4, and PXI-6. The vendor key for a Resource Manager should have an attribute for each of these specifications.

OBSERVATION: Unlike other services described in the Services Tree, there is no model key for a resource manager.

OBSERVATION: Unlike other services described in the Services Tree, no Library name is defined for a Resource Manager. This is because the Resource Manager only generates the System Description file, and provides no interface to client applications.

An example of the Services Tree is shown below. [Bracketed] lines indicate keys, with attributes indicated as <*name of attribute*> = <*value of attribute*>. Note that while Vendor A's Resource Managers implement unique sets of versions for PXI Specifications, this is not a requirement.

```
[Services]
    [Resource Managers]
        [Vendor A - Resource Manager 2.3]
            PXI-2Version = 0x00020003
            PXI-4Version = 0x00010000
            PXI-6Version = 0x00010001
        [Vendor A - Resource Manager 2.1]
            PXI-2Version = 0x00020001
            PXI-4Version = 0x00010000
            PXI-6Version = 0x00010001
        [Vendor B]
            PXI-2Version = 0x00020001
            PXI-4Version = 0x00010000
            PXI-6Version = 0x00010000
```

4.3 System Configuration File Requirements

The System Configuration File allows for the designation of the resource manager and default trigger manager for the system.

RULE: The system configuration file SHALL adhere to the same format rules described in section 2.2 for hardware description files.

RULE: The name of the default configuration file SHALL be *configuration.ini*.

RULE: On Microsoft Windows operating systems, the default configuration file SHALL be located in the `%ALLUSERSAPPDATA%\PXISA\` directory, where `%ALLUSERSAPPDATA%` is the user-independent application data folder.

RULE: Any software creating the `%ALLUSERSAPPDATA%\PXISA\` directory or adding files to it SHALL set the permissions of the directory and files to be writable by any authenticated user of the system.

PERMISSION: Any software entity or the user MAY edit the system configuration file, subject to the rules in this section.

RULE: Any software entity that changes the system configuration files SHALL open the system configuration file with exclusive access before validating and changing its configuration.

OBSERVATION: Implementation of the above rule prevents race conditions between multiple software entities attempting to edit the system configuration file simultaneously. The term *exclusive access* refers to exclusive file access mechanisms exposed by the operating system.

RULE: A software Resource Manager's installation software SHALL NOT overwrite or remove an existing system configuration file.

PERMISSION: A software Resource Manager or its installation software MAY create the system configuration file if it does not exist.

4.3.1 Resource Manager Descriptor

The resource manager descriptor defines which resource manager is responsible for generating the system description file.

RULE: The system configuration file SHALL contain at most a single resource manager descriptor.

OBSERVATION: It is possible for there to be no resource manager descriptor. For example, a resource manager may not have been selected yet, or the resource manager that was previously selected may have been uninstalled.

RULE: The Resource Manager descriptor `.ini` section header SHALL be named *ResourceManager*.

RULE: The Resource Manager descriptor SHALL contain one of each tag line types described in Table 4-1.

Table 4-1. System Configuration File – Resource Manager Tag Line Descriptions

Tag	Valid Values	Description
Name	A string indicating the name of the active Resource Manager, or the string *None*.	The name of the active resource manager for the system.
Method	A string defining either "User" or "Resource Manager".	The mechanism through which the Name tag was last set.

Resource Manager Descriptor Example

```
[ResourceManager]
Name = "PXISA Resource Manager"
Method = "Resource Manager"
```

RULE: If a software Resource Manager or its installation software creates or modifies the Resource Manager descriptor, it SHALL set the value of the Method tag in the resource manager descriptor to Resource Manager.

RULE: A software Resource Manager SHALL NOT create or modify the system description file (that is, `pxisys.ini`) unless the Name of that Resource Manager is the current Name tag value in the Resource Manager Descriptor, or no resource manager descriptor exists.

OBSERVATION: The intent of the above rule is that no software other than the active Resource Manager may modify the system description file (that is, `pxisys.ini`). This applies not only to other Resource Managers, but to other software running on the system.

PERMISSION: Any software MAY read the system description file (that is, `pxisys.ini`).

RULE: Before writing the system description file (that is, `pxisys.ini`), a Resource Manager SHALL lock the system configuration file (that is, `configuration.ini`) for exclusive access and validate that it is still the active resource manager.

RULE: A Resource Manager SHALL continue to hold the lock on the system configuration file (that is, `configuration.ini`) until it has either finished writing to the system description file (that is, `pxisys.ini`) or aborted its intended write to the system description file (that is, `pxisys.ini`).

RECOMMENDATION: Software SHOULD minimize the amount of time it holds the system configuration file (that is, `configuration.ini`) open while meeting the above requirements.

RULE: A Resource Manager descriptor SHALL be considered valid only if the Name tag refers to a Resource Manager in the Services Tree or the Name tag value is None.

RULE: A Resource Manager descriptor that is not valid SHALL be treated exactly the same as though no resource manager descriptor exists.

OBSERVATION: An invalid Resource Manager descriptor may result from the uninstallation of the active resource manager.

PERMISSION: A Resource Manager MAY change the value of the Name tag to its own name if system modules made by the vendor of that Resource Manager are present in the system and either the resource manager selection method is not set to User, or the Resource Manager descriptor is either not valid or does not exist.

OBSERVATION: Software that needs to be aware of changes to the system description file (that is, `pxisys.ini`) can use mechanisms provided by the operating system to be notified of changes as they occur.

OBSERVATION: In a system with system modules from a single vendor, that vendor's resource manager may automatically take control of the system, allowing the system to function out of the box without explicit configuration.

OBSERVATION: In systems where there are system modules from multiple vendors, it is possible that Resource Managers will compete continuously to be the active resource manager. In this case, users can set the Vendor tag value to indicate the resource manager that they want to be active, then set the Method tag to User. This can be done manually, or through a vendor-provided utility, and will result in the selection of the desired resource manager until the user chooses to modify it at a later time.

RECOMMENDATION: A resource manager vendor SHOULD provide a utility to allow the user to select the active resource manager from among those available on the system.

RECOMMENDATION: When a systems integrator or user explicitly chooses a resource manager, the configuration software SHOULD set the Method tag to User on behalf of the user.

RULE: Software SHALL NOT set the Method tag to "User" unless a user has explicitly chosen a resource manager.

OBSERVATION: A user may specify None as the value for the Vendor tag and User as the value for the Method tag to prevent a resource manager from running. This is useful primarily if the user is using a static system description file (that is, `pxisys.ini`) that resource manager software does not write.

OBSERVATION: Some contents of the system description file (that is, `pxisys.ini`) are generated in a vendor-specific way or require user input. The vendor of a Resource Manager may pull in any such data produced by another Resource Manager, which avoids unnecessarily resetting hardware configuration information, and prevents the user from having to provide the same input multiple times. For example, this technique could be used to ensure that chassis numbers are unchanged when switching between active Resource Managers, or to maintain the identification of PXI chassis for which the user was required to provide input.

OBSERVATION: A particular vendor's configuration utility will require that the Resource Manager of that vendor be the active Resource Manager to perform hardware configuration changes that impact the system description file (that is, `pxisys.ini`). For example, if a utility from vendor X receives a request from the user to set a chassis number, it must first set vendor X's Resource Manager as the active Resource Manager. However, before changing the Resource Manager, it should ask the user for approval in accordance with the rules above. In this case, the value for the Method tag should be "User".

Some systems may have two system description files, one for PXI (that is, `pxisys.ini`) and one for PXI Express (that is, `pxiesys.ini`). Refer to *PXI-6: PXI Express Software Specification* for more details. The remainder of this section applies only to systems with both system description files.

RULE: The Resource Manager SHALL NOT have both system description files open for exclusive access simultaneously.

OBSERVATION: If the Resource Manager were to attempt to have both system description files open with exclusive access simultaneously, a client opening the files in the opposite order could cause a deadlock.

RECOMMENDATION: Clients requiring consistency between both system description files SHOULD do whatever is necessary to ensure that each file's contents are synchronized with the other.

OBSERVATION: A Resource Manager can write system description files in any order, at different times. Clients that require consistency between these files can acquire a shared lock on the system configuration file (`configuration.ini`) while reading the system description files. While a client holds this lock, it is assured that the resource manager is not currently updating either system description file, and the client is guaranteed consistency.

4.3.2 Trigger Manager Descriptor

The trigger manager descriptor can be used to select a particular trigger manager as the system default, which allows trigger manager support for chassis that pre-date PXI-9. This allows vendor-specific reservation mechanisms to transition to the trigger manager while maintaining support for older chassis. Note that for chassis that have specific trigger managers indicated in the Services Tree, either for their specific model or their vendor, the system configuration file's trigger manager descriptor is irrelevant.

RULE: The system configuration file SHALL contain exactly one trigger manager descriptor.

OBSERVATION: Because the resource manager may create the system configuration file or select the Trigger Manager, it is possible not to have a trigger manager descriptor if the Resource Manager has not yet run. Also, a Trigger Manager may not be installed on the system, which would prevent a selection from being made.

RULE: The Trigger Manager descriptor `.ini` section header SHALL be named *TriggerManager*.

RULE: The Trigger Manager descriptor SHALL contain one of each tag line type described in Table 4-2.

Table 4-2. System Configuration File—Trigger Manager Tag Line Descriptions

Tag	Valid Values	Description
Vendor	A string indicating the vendor name for the default Trigger Manager or "None" if no Trigger Manager is available.	The default trigger manager for the system.
Method	A string defining either User or Resource Manager.	The mechanism through which the Vendor tag was last set.

Trigger Manager Descriptor Example

```
[TriggerManager]
Vendor = "PXISA"
Method = "Resource Manager"
```

RULE: If a software Resource Manager or its installation software creates or modifies the Trigger Manager descriptor, it SHALL set the value of the Method tag in the trigger manager descriptor to Resource Manager.

RULE: A Trigger Manager descriptor SHALL be considered valid only if the Vendor tag refers to a vendor that has registered vendor default Trigger Manager in the Services Tree.

OBSERVATION: The default trigger manager may be chosen from any vendor-default trigger manager in the Services Tree.

RULE: An invalid Trigger Manager descriptor SHALL be treated exactly the same as if no Trigger Manager descriptor exists.

RULE: A Resource Manager SHALL NOT modify the Trigger Manager descriptor if the value of the Method tag is User and the Trigger Manager descriptor is valid.

RULE: If the Trigger Manager descriptor is invalid or does not exist, the Resource Manager SHALL set the Vendor tag to a valid value of its choosing before writing the system description file (that is, `pxisys.ini`), assuming a Trigger Manager that can act as a default exists on the system.

RULE: The Resource Manager SHALL set the Vendor tag to "None" before writing the system description file (that is, `pxisys.ini`) if no default Trigger Manager is available in the Services Tree.

PERMISSION: A Resource Manager MAY change the value of the Vendor tag to any valid value if it is the selected Resource Manager in the Resource Manager descriptor, and either the Method tag in the Trigger Manager descriptor is "Resource Manager" or the Trigger Manager selection is invalid.

OBSERVATION: The active resource manager generally chooses the default trigger manager unless a user explicitly sets it.

RECOMMENDATION: A resource manager vendor SHOULD provide a utility to allow the user to select the default trigger manager from among those available on the system.

RECOMMENDATION: When a systems integrator or user explicitly chooses a Trigger Manager, the configuration software SHOULD set the Method tag to "User" on behalf of the user.

RULE: Software SHALL NOT set the Method tag to "User" unless the user has explicitly chosen a default Trigger Manager.