PXI™-9

PXI and PXI Express Trigger Management Specification

PCI eXtensions for Instrumentation

PCI EXPRESS eXtensions for Instrumentation

Revision 1.1
May 31, 2018

PXI Systems Alliance

IMPORTANT INFORMATION

Copyright

NOTICE

The *PXI and PXI Express Trigger Management Specification* is authored and copyrighted by the PXI Systems Alliance. The intent of the PXI Systems Alliance is for the *PXI and PXI Express Trigger Management 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 trademark 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 and PXI Express Trigger Management Specification Revision History

This section is an overview of the revision history of the *PXI and PXI Express Trigger Management Specification*.

Revision 1.0, October 18, 2012

This is the first public revision of the *PXI and PXI Express Trigger Management Specification*.

Revision 1.1, May 31, 2018

Added text regarding coordination of multivendor static trigger management.

Clarified some error handling language.

Updated text regarding versioning of Trigger Managers in the Services Tree.

Added support for the Linux operating system.

This Page Intentionally Left Blank

Contents

1. Introduction

1.1 Objectives 1
1.2 Intended Audience and Scope 2
1.3 Background and Terminology 2
1.4 Applicable Documents 3

2. PXI Trigger Management Software Services

2.1 Overview 5
2.2 PXI Trigger Manager 5
2.2.1 Theory of Operations 5
2.2.2 PXISA_ChassisTrig_OpenChassis 6
2.2.3 PXISA_ChassisTrig_CloseChassis 8
2.2.4 PXISA_ChassisTrig_SetReservation 8
2.2.5 PXISA_ChassisTrig_SetReservationMultiple 10
2.2.6 PXISA_ChassisTrig_SetRoute 11
2.2.7 PXISA_ChassisTrig_ClearRoute 12
2.2.8 PXISA_ChassisTrig_GetLineInformation 12
2.2.9 PXISA_ChassisTrig_ClearAllRoutesAndReservations 13
2.3 Backward Compatibility 14
2.4 Status Codes 14
2.5 Registration of Trigger Managers 15
2.5.1 Trigger Manager Services Tree 15

3. Software Frameworks and Requirements

3.1 Overview 19
3.2 32-bit Windows System Framework 19
3.2.1 Introduction 19
3.2.2 Trigger Manager Software Bindings 19
3.2.3 Services Tree Implementation 20
3.3 32-Bit Linux System Framework 20
3.3.1 Introduction 20
3.3.2 Trigger Manager Software Bindings 20
3.3.3 Service Tree Implementation 20
3.3.4 Security of PXI Files and Interfaces 21
3.4 64-Bit Windows System Framework 21
3.4.1 Introduction 21
3.4.2 Trigger Manager Software Bindings 21
3.4.3 Services Tree Implementation 22
3.5 64-Bit Linux System Framework 22
3.5.1 Introduction 22
3.5.2 Trigger Manager Software Bindings 22
3.5.3 Service Tree Implementation 23
3.5.4 Security of PXI Files and Interfaces 23

Appendix A: Trigger Manager API Example Code

Setting and Clearing a Reservation 25
Reserving Multiple Lines Simultaneously 25
Setting a Route 26
Clearing a Route 27
Printing Line State 27

Appendix B: 32-Bit and 64-Bit Windows System Framework Files and Client Example

PXITriggerManager.h 29
PXITriggerManager.def 32
Client Example 32

1. Introduction

This section explains the objectives and scope of the *PXI and PXI Express Trigger Management Specification*. It also describes the intended audience and lists relevant terminology and documents. Note that this specification is intended to supplement the *PXI Software Specification* and *PXI Express Software Specification*. Refer to the *PXI Software Specification* and *PXI Express Software Specification* for software information not directly related to the PXI Trigger Manager.

1.1 Objectives

The *PXI and PXI Express Trigger Management Specification* provides a standard for the management of trigger bus resources described in the *PXI Hardware Specification* and *PXI Express Hardware Specification*.

There are five major objectives for the *PXI and PXI Express Trigger Management Specification*.

The first objective is to define software interfaces for the reservation of trigger bus resources. *PXI-1: PXI Hardware Specification* and *PXI-5: PXI Express Hardware Specification* describe the PXI trigger bus for intermodule synchronization and communication.The trigger bus can be driven by multiple peripherals simultaneously, but doing so can result in hardware damage. To prevent such double driving, the *PXI and PXI Express Trigger Management Specification* defines a software interface through which clients can reserve these lines to be used. Responsibility is with clients to reserve these trigger lines before using them. Conformance to this requirement guarantees that a single trigger line cannot be driven by multiple clients simultaneously. The interfaces defined in *PXI-9: PXI and PXI Express Trigger Management Specification* work in concert with mechanisms described in *PXI-2: PXI Software Specification* and *PXI-6: PXI Express Software Specification*, and clients of the Trigger Manager API will need to be familiar with those specifications.

The second objective is to define software interfaces for the routing of signals across a trigger bus bridge. The *PXI Hardware Specification* allows for chassis with multiple trigger bus segments, and some such chassis are designed with a trigger bus bridge to allow the transmission of signals between trigger bus lines. The interfaces defined in *PXI-9: PXI and PXI Express Trigger Management Specification* allow client software to operate these trigger bridges without using vendor-specific interfaces. These interfaces work in concert with mechanisms described in *PXI-2: PXI Software Specification* and *PXI-6: PXI Express Software Specification*, and clients of the Trigger Manager API will need to be familiar with those specifications.

The third objective is compatibility with previous PXI hardware and PXI software specifications. The *PXI and PXI Express Trigger Management Specification* imposes no new requirements on PXI and PXI Express hardware. The software interface described allows for the use of hardware capabilities that may or may not exist in a particular PXI or PXI Express chassis, but hardware that does not support these capabilities remains in full compliance with this specification. Additionally, like previous software specifications, the *PXI and PXI Express Trigger Management Specification* can be implemented independently of any additional software such as instrument drivers.

The fourth objective is to provide a standardized mechanism for the implementation of trigger routing and reservation APIs predating this specification. Interfaces defined in the VISA specifications by the IVI Foundation provide mechanisms to access trigger resources in PXI and PXI Express chassis. This specification aims to allow these interfaces to be implemented in a way that functions across multiple vendors seamlessly.

The fifth objective is to define standard operating system frameworks. This includes support of standard operating systems such as Microsoft Windows.

1.2 Intended Audience and Scope

This specification is primarily intended for product developers interested in implementing and leveraging trigger management features of the PXI and PXI Express platforms. Software developers and systems integrators should take advantage of these software interfaces to manage PXI and PXI Express trigger resources. Additionally, product developers and systems integrators should reference the operating system framework definitions to ensure system-level interoperability. Note that the definitions and interfaces described in this document apply to both PXI and PXI Express hardware components.

1.3 Background and Terminology

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

- **API**—Application Programming Interface
- **PCI**—Peripheral Component Interconnect; electrical specification defined by PCISIG
- **PCISIG**—PCI Special Interest Group
- **PXI**—PCI eXtensions for Instrumentation
- **VISA**—IVI specifications VPP-4.3 and related

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 PXI and PXI Express specifications referenced in this section. It is assumed that the reader has a thorough understanding of PXI and PXI Express. The PXI and PXI Express specifications refer 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*
- *PXI-2: PXI Software Specification*
- *PXI-5: PXI Express Hardware Specification*
- *PXI-6: PXI Express Software Specification*

This Page Intentionally Left Blank

2. PXI Trigger Management Software Services

This section defines the PXI and PXI Express Software Services for Trigger Management, their APIs, their registration, and how they interact with clients of PXI and PXI Express Trigger Resources.

2.1 Overview

This section defines the Trigger Manager service that shall be provided for each chassis. It further defines how this service should be registered and how clients in the system use it.

The APIs and databases defined in this section are described in a platform-independent manner. The platform-specific details of this specification are found in Section 3, *Software Frameworks and Requirements*.

2.2 PXI Trigger Manager

A PXI Trigger Manager is responsible for:

- Providing reservation functionality for each trigger bus line in the chassis for which it is registered.
- Providing routing functionality between trigger bus lines, where such routing functionality is supported by chassis hardware.
- Enforcing the specific routing capabilities of a chassis by returning well-defined error codes when unsupported or invalid requests are made.
- Facilitating the guarantee of exclusive access to drive each individual trigger for a single client at any point in time.

2.2.1 Theory of Operations

Exclusive access to trigger lines is accomplished through a cooperative interaction with all clients. A client may open one or more sessions on a particular Trigger Manager for a particular chassis. Each session is associated with a label the client provides when the session is opened, and all operations performed with that session are bound to, or “owned” by, that label. Clients must then be implemented to reserve a line with the appropriate Trigger Manager before using hardware to drive a signal on that line. A Trigger Manager relies on adherence to this policy to prevent one client from inadvertently interfering with another client’s exclusive use of a particular line.

Once a client has reserved a particular line, that client may use the associated Trigger Manager to route a signal to that line from another trigger line if such functionality is supported by the hardware. Because this process involves driving a signal to the destination line, the client must have a reservation for that line before the Trigger Manager allows the route, and that reservation must be maintained until the route is cleared. In this case, the client is expected not to configure hardware to drive the destination line directly, because this would potentially conflict with the signal being driven by the trigger bridge, resulting in hardware damage.

A client determines the appropriate Trigger Manager for a chassis by accessing a key in the services tree, as directed by information in the system description files. The client can then open one or more sessions to one or more chassis, manipulating reservations and routes as necessary. For an example detailing this algorithm, refer to *Appendix B*: *32-Bit and 64-Bit Windows System Framework Files*.

A PXI Trigger Manager must implement the operations listed in this section to the extent that the relevant chassis supports them. The operations are specified here by their names, return values, input parameters, and output parameters. Output parameters are differentiated by an asterisk in the parameter list. Although this corresponds roughly to the C language notation for passing a pointer, it does not necessarily indicate that a

pointer is used in the implementation. Arrays are differentiated by brackets and an ellipsis ([...]) after the parameter name.

RULE: A Trigger Manager SHALL implement all operations described in this section.

PERMISSION: A Trigger Manager implementation MAY return `kPXISA_ErrorUnsupported` in response to `PXISA_ChassisTrig_SetRoute` and `PXISA_ChassisTrig_ClearRoute` operations if the chassis hardware does not support routing.

RULE: When returning a negative error code as its status, a method of a Trigger Manager SHALL NOT make any change to routes or reservations configured in the system.

RULE: When returning a positive warning code or 0 as its status, a method of a Trigger Manager SHALL successfully perform the operation requested by the caller.

RULE: The installation software for a PXI or PXI Express chassis SHALL include a Trigger Manager implementation that supports the chassis.

OBSERVATION: Because a single Trigger Manager can be designated for multiple chassis as described in section 2.5, a vendor can satisfy the above rule for any number of chassis by creating only one Trigger Manager.

OBSERVATION: A client of the Trigger Manager must treat output parameters of a Trigger Manager operation as undefined if the operation returns an error status code, unless otherwise specified.

2.2.2 PXISA_ChassisTrig_OpenChassis

Allows a particular client to open a session to a chassis to be used in subsequent calls to other methods on the trigger manager. Ownership is established for the session such that all operations made with that session have the same owner as the session used to perform them.

```
Status PXISA_ChassisTrig_OpenChassis(Integer chassisNum, String clientLabel,
Session * session)
```

chassisNum: The chassis number of the chassis to open from the System Description File.

clientLabel: A label identifying the client that owns this session.

session (out): A session identifier to be used in subsequent calls.

RULE: A Trigger Manager SHALL return `kPXISA_ErrorInvalidParameter` in response to a call to `PXISA_ChassisTrig_OpenChassis` if the `clientLabel` is something other than a string of nonzero length.

RULE: A Trigger Manager SHALL return `kPXISA_ErrorInvalidParameter` in response to a call to `PXISA_ChassisTrig_OpenChassis` if `chassisNum` is not a valid chassis number from the PXI or PXI Express System Description File.

RULE: If a Trigger Manager contains a cache of the chassis numbers and/or the physical chassis hardware they represent, the Trigger Manager SHALL ensure that the cache is up to date when `PXISA_ChassisTrig_OpenChassis` is invoked.

RULE: A Trigger Manager SHALL allow multiple sessions to be opened simultaneously regardless of client label.

RULE: A Trigger Manager SHALL allow multiple sessions to be opened simultaneously to the same physical chassis.

RULE: A Trigger Manager SHALL prevent a session opened with a given client label from modifying configuration changes made by any session opened with a different client label.

RULE: A Trigger Manager SHALL NOT enforce any distinction in ownership between the configuration changes made by one session and the configuration changes made by a different session where both sessions were opened with the same client label.

OBSERVATION: A session returned from `PXISA_ChassisTrig_OpenChassis` is intended to be opaque to clients. Clients should not assume that sessions can be meaningfully used in numeric comparisons or other operations outside of the Trigger Manager.

RULE: All routes and reservations made with the session returned by `PXISA_ChassisTrig_OpenChassis` SHALL be "owned" by every client that passed an identical `clientLabel` to `PXISA_ChassisTrig_OpenChassis`.

OBSERVATION: The client label establishes a basis for ownership of routes and reservations by a particular client. The level of granularity enforced here is up to the client; different instances of the same client may use unique client labels, or they may use the same client label if the author of the application desires that all instances be treated as a single entity. In any case, the intention is that no caller may modify the configuration changes made by another caller unless the same client label has been used by both callers.

OBSERVATION: A client of the Trigger Manager API can prevent its client label from being confused with the client label of any other client by using a Globally Unique Identifier (GUID) as all or part of its client label.

OBSERVATION: The clientLabel string provided by the client can aid the user in determining the source of reservations and routes. It is thus to the user's advantage for clients to choose a descriptive clientLabel. For instance, including the reserving software's vendor's name in the client label may be helpful to the user.

OBSERVATION: `PXISA_ChassisTrig_OpenChassis` can be used to perform any initialization functionality necessary for the internal workings of a Trigger Manager.

RULE: A Trigger Manager SHALL NOT reassign reservations or routes created on one physical chassis to another physical chassis as a result of configuration changes in the system.

PERMISSION: A Trigger Manager MAY maintain reservations and routes for a physical chassis over a configuration change that does not change the chassis's hardware or physical identity, such as a simple renumbering of a chassis.

OBSERVATION: Because the chassis number is used only in the `PXISA_ChassisTrig_OpenChassis` function, it is possible to create a Trigger Manager implementation in which routes and reservations can be maintained on a particular physical chassis even when the chassis number of that physical chassis changes one or more times. It is also possible to maintain validity of sessions open to such a chassis regardless of changes to the chassis number.

PERMISSION: A Trigger Manager MAY clear all routes and reservations for a chassis whose chassis number has changed.

RULE: A Trigger Manager that implements the above permission SHALL treat as disconnected all open sessions referring to a chassis whose chassis number has changed.

RULE: A Trigger Manager SHALL treat as disconnected all open sessions referring to chassis that are no longer present.

RULE: A Trigger Manager SHALL return `PXISA_ErrorDisconnected` when a client attempts to use a disconnected session.

RULE: A Trigger Manager SHALL NOT reuse any disconnected session handles until those sessions are explicitly closed by the client.

OBSERVATION: A client is still required to close disconnected sessions, in order to allow the Trigger Manager to free resources associated with the session.

2.2.3 PXISA_ChassisTrig_CloseChassis

Closes a previously active session.

```
PXISA_ChassisTrig_CloseChassis(Session session)
```

session: The session to close.

RULE: A call to `PXISA_ChassisTrig_CloseChassis` SHALL NOT make any modifications to routes or reservations configured in the system.

OBSERVATION: The above rule is intended to enforce the maintenance of routes and reservations between sessions. This allows a client application to create routes and reservations and be guaranteed that they will be enforced until the system is rebooted or they are intentionally cleared.

OBSERVATION: `PXISA_ChassisTrig_CloseChassis` has no return value because there is no reasonable action to take on failure. It should always succeed.

OBSERVATION: `PXISA_ChassisTrig_CloseChassis` can be used to perform any clean-up operations associated with the internal implementation of a trigger manager.

OBSERVATION: Failure to close a trigger manager session may result in memory or other resource leaks.

OBSERVATION: `PXISA_ChassisTrig_CloseChassis` has no return type.

2.2.4 PXISA_ChassisTrig_SetReservation

Changes the reservation state of a line.

```
Status PXISA_ChassisTrig_SetReservation(Session session, Integer bus, Integer
line, Integer reserve)
```

session: The session that identifies the chassis and client for which the reservation state should be modified.

bus: The number of the trigger bus for which the reservation state should be modified.

line: The 0-indexed line for which the reservation state should be modified.

reserve: The desired reservation state, from the following table.

reserve Value	Definition
0	Clear the line's reservation.
1	Reserve the line.

RULE: A Trigger Manager SHALL return `kPXISA_ErrorInvalidParameter` if a call to `PXISA_ChassisTrig_SetReservation` passes a bus number that is not a valid trigger bus number from the system description file.

RULE: A Trigger Manager SHALL return `kPXISA_ErrorInvalidParameter` if a call to `PXISA_ChassisTrig_SetReservation` passes an invalid line number as the line parameter.

RULE: A Trigger Manager SHALL return `kPXISA_ErrorInvalidClient` in response to a call to `PXISA_ChassisTrig_SetReservation` if the provided bus and line correspond to a line reserved by a client other than the owner of the session requesting the change.

RULE: A Trigger Manager SHALL return `kPXISA_ErrorConflictingRoute` if a call to `PXISA_ChassisTrig_SetReservation` attempts to clear the reservation of a line reserved by the same client making the request, but which is also the destination of a route.

OBSERVATION: A client must clear a route before clearing any reservation that was required to set that route.

RULE: A Trigger Manager SHALL return `kPXISA_ErrorLineAlreadyReserved` if a call to `PXISA_ChassisTrig_SetReservation` attempts to reserve a line that is already reserved by the client making the request.

RULE: A Trigger Manager SHALL return `kPXISA_ErrorLineNotReserved` if a call to `PXISA_ChassisTrig_SetReservation` attempts to clear the reservation of a line that is not reserved.

RULE: A Trigger Manager SHALL return `kPXISA_ErrorInvalidParameter` if a call to `PXISA_ChassisTrig_SetReservation` passes as the reserve value anything other than the values listed in the table above.

RULE: A Trigger Manager SHALL maintain the reservation of lines reserved with `PXISA_ChassisTrig_SetReservation` or `PXISA_ChassisTrig_SetReservationMultiple` in its internal state until they are cleared by a call to `PXISA_ChassisTrig_SetReservation` with a session having the appropriate client label, or until the system is rebooted.

RULE: A Trigger Manager SHALL NOT maintain the reservation of lines reserved with `PXISA_ChassisTrig_SetReservation` across system reboots.

OBSERVATION: PXI trigger lines are a limited resource in the chassis. Each client is expected to unreserve lines that are no longer required in order to prevent unnecessary consumption of trigger lines.

In accordance with the previous OBSERVATION, it should typically be the case that an application that will use trigger lines will set the required reservations, execute the application logic that requires those reservations, and then clear the reservations when they are no longer needed. However, a vendor may choose to provide a tool that creates reservations that are persisted indefinitely, including across reboots, until a user has explicitly cleared them. These are known as *Static Reservations*.

OBSERVATION: A tool providing static reservations must re-apply its settings to Trigger Manager(s) when the system boots.

RECOMMENDATION: A tool implementing static reservations SHOULD NOT obtain a reservation at boot that it did not already hold during the previous system shutdown.

OBSERVATION: If multiple static reservation tools are provided by multiple vendors, collisions can occur if those tools attempt at boot to reserve overlapping sets of trigger lines. If all such tools adhere to the above RECOMMENDATION, conflicts between them will never occur.

OBSERVATION: There is no guarantee that a tool providing static reservations will not collide with another application which requests conflicting non-static reservations upon its execution. Such an application may be manually launched by the user shortly after boot, or configured to start automatically. To avoid this, a tool providing static reservations should be designed to perform its startup reservation requests as early as possible at system boot.

2.2.5 PXISA_ChassisTrig_SetReservationMultiple

Changes the reservation state of multiple lines simultaneously. The index of failure parameter is optional, because the client may not need this information.

```
Status PXISA_ChassisTrig_SetReservationMultiple(Session session, Integer
numElements, Integer buses [...], Integer lines [...], Integer * indexOfFailure)
```

session: The session that identifies the chassis and client for which the reservation state should be modified.

numElements: The number of bus/line pairs for which to change the reservation state.

buses [...]: An array of `numElements` trigger bus numbers on which to reserve lines. The *n*th element of `buses` and *n*th element of `lines` together make a single reservation request.

lines [...]: An array of `numElements` 0-indexed lines on `buses`. The *n*th element of `buses` and *n*th element of `lines` together make a single reservation request.

indexOfFailure (opt,out): The 0-based index of the value in each of `buses` and `lines` for which the reservation could not be changed, if an error was returned.

RULE: A Trigger Manager SHALL execute a call to `PXISA_ChassisTrig_SetReservationMultiple` by considering each bus/line pair expressed in the `buses` parameter and the `lines` parameter as a single reservation to be set.

RULE: A Trigger Manager SHALL execute a call to `PXISA_ChassisTrig_SetReservationMultiple` atomically with respect to other requests for reservations.

RULE: A Trigger Manager SHALL return errors on a call to `PXISA_ChassisTrig_SetReservationMultiple` in a manner identical to that described for `PXISA_ChassisTrig_SetReservation` in section 2.2.8, applying all rules to each individual reservation to be set.

RULE: A Trigger Manager SHALL return `kPXISA_ErrorInvalidParameter` if a call to `PXISA_ChassisTrig_SetReservationMultiple` provides a negative value for the `numElements` parameter.

RULE: A Trigger Manager SHALL return `kPXISA_ErrorInvalidParameter` if a call to `PXISA_ChassisTrig_SetReservationMultiple` lists the same bus/line pair twice.

RULE: A Trigger Manager SHALL maintain the reservation of lines reserved with `PXISA_ChassisTrig_SetReservationMultiple` according to the rules described for `PXISA_ChassisTrig_SetReservation` above.

RULE: If the operation requested in a call to `PXISA_ChassisTrig_SetReservationMultiple` produces an error for any of the bus/line pairs provided, the Trigger Manager SHALL NOT alter the state of the system.

RULE: If the operation requested in a call to `PXISA_ChassisTrig_SetReservationMultiple` completes successfully, the Trigger Manager SHALL return –1 in the `indexOfFailure` parameter.

RULE: If the operation requested in a call to `PXISA_ChassisTrig_SetReservationMultiple` produces an error for any of the bus/line pairs provided, the Trigger Manager SHALL return the index of a bus/line pair that caused the error in the `indexOfFailure` parameter.

OBSERVATION: The `indexOfFailure` parameter informs the caller of only a single error that occurred. It is possible that multiple bus/line pairs in the provided lists cannot be reserved, but the client must make multiple requests to run into these problems and understand their reasons.

OBSERVATION: Because other clients may be actively changing the reservation state of the lines, it is possible that a reservation that fails in one call will succeed in another call a few moments later, or that a reservation that succeeds in one call will fail in another call a few moments later.

OBSERVATION: Clients that encounter failures using `PXISA_ChassisTrig_SetReservationMultiple` due to trigger bus resource conflicts with other clients can combine the information provided by the `indexOfFailure` parameter with the description of the chassis trigger resources in the system description file to resolve the conflicts. If a conflict is encountered, an application can iteratively attempt the reservation of acceptable resource combinations until it finds a combination that is available.

OBSERVATION: Race conditions can occur between multiple clients trying to reserve sets of lines when those reservations are performed separately. When a related set of lines is required, for example, to route a signal across multiple segments of a chassis, a client can eliminate a class of race conditions by requesting reservations simultaneously using `PXISA_ChassisTrig_SetReservationMultiple`.

OBSERVATION: A tool reapplying multiple static reservations at boot can do so more efficiently with `PXISA_ChassisTrig_SetReservationMultiple` than by multiple calls to `PXISA_ChassisTrig_SetReservation`.

2.2.6 PXISA_ChassisTrig_SetRoute

Provides a mechanism to set a route across a trigger bridge on a chassis backplane. Because the existence of a route requires the use of the destination trigger line, the client is responsible for reserving that line prior to making this call.

```
Status PXISA_ChassisTrig_SetRoute(Session session, Integer sourceBus, Integer
sourceLine, Integer destBus, Integer destLine)
```

session: The session that identifies the chassis and client for which the route is made.

sourceBus: The number of the trigger bus that is the source of the signal to be routed.

sourceLine: The 0-indexed line that is the source of the signal to be routed.

destBus: The number of the trigger bus that is the destination of the signal to be routed.

destLine: The 0-indexed line that is the destination of the signal to be routed.

RULE: A Trigger Manager SHALL return `kPXISA_ErrorUnsupported` if a call to `PXISA_ChassisTrig_SetRoute` requests a route that the hardware is not physically capable of making.

RULE: A Trigger Manager SHALL return `kPXISA_ErrorInvalidParameter` if a call to `PXISA_ChassisTrig_SetRoute` passes as `sourceBus` or `destBus` a number that is not a valid trigger bus number from the system description file.

RULE: A Trigger Manager SHALL return `kPXISA_ErrorInvalidParameter` if a call to `PXISA_ChassisTrig_SetRoute` passes an invalid line number as the `sourceLine` or `destLine` parameter.

RULE: A Trigger Manager SHALL require that a trigger line be reserved by the client prior to being used as a destination line in a call to `PXISA_ChassisTrig_SetRoute`. If the line is already reserved by a client other than the client calling `PXISA_ChassisTrig_SetRoute`, or if the line is not reserved by any client, a Trigger Manager SHALL return `kPXISA_ErrorLineNotReserved`.

RULE: A Trigger Manager SHALL return `kPXISA_ErrorConflictingRoute` if a call to `PXISA_ChassisTrig_SetRoute` specifies a destination bus and line that are already the destination of a route made previously by the same client.

RULE: A Trigger Manager SHALL maintain routes set with `PXISA_ChassisTrig_SetRoute` in the hardware state until they are cleared by a call to `PXISA_ChassisTrig_ClearRoute` or `PXISA_ChassisTrig_ClearAllRoutesAndReservations` with a session having the appropriate client label, or until the system is rebooted.

RULE: A Trigger Manager SHALL NOT maintain routes set with `PXISA_ChassisTrig_SetRoute` across system reboots.

OBSERVATION: The destination line of a route must be reserved by the caller to guarantee that the destination is not also driven by a device controlled by another client.

OBSERVATION: Setting a route has no impact or requirement on the reservation of the source bus and source line.

A vendor may choose to provide a tool which creates routes that are persisted indefinitely, including across reboots, until a user has explicitly cleared them. These are known as *Static Routes*.

OBSERVATION: A static route will always require a corresponding static reservation of its destination bus.

OBSERVATION: All OBSERVATIONS and RECOMMENDATIONS pertaining to static reservations in section 2.2.4 also apply to static routes.

2.2.7 PXISA_ChassisTrig_ClearRoute

Clears a route previously set by the client.

```
Status PXISA_ChassisTrig_ClearRoute(Session session, Integer destBus, Integer destLine)
```

session: The session that identifies the chassis and client for which the route will be cleared.

destBus: The number of the trigger bus that is the destination of the route to be cleared.

destLine: The 0-indexed line that is the destination of the route to be cleared.

RULE: A Trigger Manager SHALL return `kPXISA_ErrorInvalidParameter` in response to a call to `PXISA_ChassisTrig_ClearRoute` if the provided destination bus and line do not correspond to a route previously set by a call to `PXISA_ChassisTrig_SetRoute`.

RULE: A Trigger Manager SHALL return `kPXISA_ErrorInvalidClient` in response to a call to `PXISA_ChassisTrig_ClearRoute` if the provided destination bus and line corresponds to a route owned by a client other than the owner of session.

OBSERVATION: Clearing a route has no impact on the reservation of the destination line. If a client is finished with the line, it is expected to clear the reservation with a separate call to `PXISA_ChassisTrig_SetReservation`.

2.2.8 PXISA_ChassisTrig_GetLineInformation

Provides a mechanism for a caller to obtain the reservation, routing, and ownership state of a given line. All output parameters except the reservation state are optional, in case the client does not need that information.

```
Status PXISA_ChassisTrig_GetLineInformation(Session session, Integer bus, Integer line, Integer * reserve, Integer * routeSrcBus, Integer * routeSrcLine, String * clientLabel)
```

session: The session that identifies the chassis and client for which to retrieve information.

bus: The number of the trigger bus for which the state should be returned.

line: The 0-indexed line for which the state should be returned.

reservation (out): The current line reservation state, from the following table.

routeSrcBus (opt,out): The source bus for the route, if the line is routed.

routeSrcLine (opt,out): The source line for the route, if the line is routed.

owner (opt,out): The client label for the owner of the reservation, if the line is reserved.

reservation Value	Definition
0	The line is not reserved.
1	The line is reserved, and is not the destination of a route.
2	The line is reserved, and is the destination of a route.

RULE: A Trigger Manager SHALL return `kPXISA_ErrorInvalidParameter` if a call to `PXISA_ChassisTrig_GetLineInformation` passes a bus number that is not a valid trigger bus number from the system description file.

RULE: A Trigger Manager SHALL return `kPXISA_ErrorInvalidParameter` if a call to `PXISA_ChassisTrig_GetLineInformation` passes an invalid line number as the line parameter.

RULE: A Trigger Manager SHALL provide one of the values from the table above for the reserve parameter.

OBSERVATION: It is possible for a caller to use `PXISA_ChassisTrig_GetLineInformation` to query the owning client for a particular route or reservation, then open a new session to impersonate that client and modify the route or reservation.

OBSERVATION: If the reservation value returned indicates the line is not reserved, the owner parameter is undefined, and the caller should not access it.

2.2.9 PXISA_ChassisTrig_ClearAllRoutesAndReservations

Provides the capability to clear all routes and reservations for a particular client on a given chassis.

```
Status PXISA_ChassisTrig_ClearAllRoutesAndReservations(Session session)
```

session: The session that identifies the chassis and client for which to clear all configuration.

RULE: A Trigger Manager SHALL clear all the routes and reservation for the chassis and client associated with the specified session when `PXISA_ChassisTrig_ClearAllRoutesAndReservations` is called.

RULE: A Trigger Manager SHALL NOT, as a result of a call to `PXISA_ChassisTrig_ClearAllRoutesAndReservations`, modify reservations or routes for chassis other than the chassis associated with the provided session.

RULE: A Trigger Manager SHALL NOT, as a result of a call to `PXISA_ChassisTrig_ClearAllRoutesAndReservations`, modify reservations or routes owned by clients other than the client associated with the provided session.

2.3 Backward Compatibility

Software interfaces specified by the IVI Foundation which predate this specification allow for the reservation of trigger lines. One aim of this specification is to allow the implementation of these interfaces such that they continue to operate on PXI and PXI Express chassis for which there is no Trigger Manager implementation. This is necessary to allow vendors to abandon vendor-specific mechanisms but still have full backward compatibility for these chassis.

RULE: A Trigger Manager implementation designated as a vendor default as described in Section 2.5.1, *Trigger Manager Services Tree*, SHALL be capable of supporting reservation requests for any PXI or PXI Express chassis defined in the system description file(s).

OBSERVATION: All chassis information necessary to fulfill the above rule is available in the system description files.

2.4 Status Codes

Many of the methods defined for the Trigger Manager API require the definition of specific status codes.

RULE: All status codes returned by a Trigger Manager between negative 1000 and positive 1000 SHALL be reserved for use by PXISA.

RULE: A status code of zero (0) SHALL be used to represent a successful operation.

RULE: A negative status code SHALL be used to represent a failure. The status code negative one (–1) is reserved by this specification to represent a generic failure.

PERMISSION: A Trigger Manager may return other values to indicate a specific type of failure, as long as those values are less than negative 1000.

RULE: A positive status code SHALL be used to represent a warning. The status code positive one (1) is reserved by this specification to represent a generic warning.

PERMISSION: A Trigger Manager may return other values to indicate a specific type of warning, as long as those values are more than positive 1000.

RULE: A Trigger Manager SHALL return the status codes in the table below in circumstances described in the *Definition* column.

Mnemonic	Status Value	Definition
`kPXISA_Warning`	1	Generic warning.
`kPXISA_Success`	0	Success.
`kPXISA_Error`	–1	Generic error.
`kPXISA_ErrorUnsupported`	–2	The hardware does not support the requested operation.
`kPXISA_ErrorInvalidParameter`	–3	An input parameter was invalid.
`kPXISA_ErrorLineNotReserved`	–4	An attempt was made to route to a line the calling client has not reserved. An attempt was made to unreserve a line that was not reserved.

Mnemonic	Status Value	Definition
kPXISA_ErrorLineAlreadyReserved	–5	An attempt was made to reserve a line already reserved by this client. Note that this status code will NOT be used if a different client already reserved the line. Refer to kPXISA_ErrorInvalidClient below.
kPXISA_ErrorConflictingRoute	–6	An attempt was made to route to a line that is already the destination of another route by this client, or to unreserve a line that is the destination of a route by this client.
kPXISA_ErrorInvalidClient	–7	A route or reservation was attempted that would require another client's route or reservation to be changed.
kPXISA_ErrorDisconnected	–8	The client attempted to use a session for which the chassis is disconnected.

2.5 Registration of Trigger Managers

The Trigger Managers are invoked by clients and specified to clients by the Resource Manager. The clients and Resource Manager need a central registry where they can find information about how to invoke the Trigger Manager. This central registry shall be the Services Tree, as defined in the *PXI Express Software Specification*. This subsection describes additions to the Services Tree required for the registration of Trigger Managers.

RULE: Trigger Managers SHALL include an installer that places references to the Trigger Manager in the Services Tree, as described in this section.

2.5.1 Trigger Manager Services Tree

The Services Tree is a hierarchical database of the services available in a PXI or PXI Express system. Each element in the Services Tree is either a key or an attribute.

Each key has:

- One name.
- One parent key (exception: the root key has no parent).
- Zero or more child keys.
- Zero or more attributes.

Each attribute has:

- One name.
- One type, which is either Integer or String.
- One value.

The root of the Services Tree is named *Services*.

The child keys of the root key are called *category keys*. A category key called *Trigger Managers* is required to allow registration of Trigger Managers.

The child keys of the category keys are called *vendor keys*. The vendor keys under the Trigger Managers category key and their descendants are created by the installation software for the Trigger Managers. The name of a vendor key is a unique string identifying the vendor of the Trigger Manager being installed.

RULE: A vendor key must match the Vendor field from the chassis description file, for the chassis vendor that it represents.

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

The child keys of the vendor keys are called model keys. The name of a model key is also the model name of a chassis for which the designated Trigger Manager should be used to access trigger line resources.

RULE: The name of each model key SHALL be unique within a vendor key.

RULE: The name of each model key shall match the Model field from the chassis description file for the chassis model it represents.

RULE: Each model key SHALL have a string attribute whose name is *Library* and whose value is the path to the library implementing the Trigger Manager for that chassis.

RULE: Each model key SHALL have an integer attribute whose name is *Version* and whose value is 0x00010000.

OBSERVATION: The value of the Version key is the major and minor version of the Trigger Manager interfaces described in this chapter, where the major version is expressed in the top 16 bits, and the minor version is expressed in the lower 16 bits.

OBSERVATION: Future versions of this specification will increment the minor version number of the interface to indicate that new capabilities have been added to the interfaces, but that backward compatibility has been maintained. The major version number will be incremented only if backward compatibility with previous versions of the interface is broken. Ideally, incrementing the major version number should be avoided in future updates to this specification. If no changes have been made to the Trigger Manager interface, the Version attribute will be unchanged.

OBSERVATION: While updates to the interface version have matched their corresponding specification versions thus far, this is done for convenience. The above RULE should be taken as the sole authority on the interface version, and the modification of the Version attribute value in the above RULE should be expected to rev independently of the revision of this specification.

OBSERVATION: A Trigger Manager client must use the value of the Version attribute to identify the version of the Trigger Manager interface the Trigger Manager interface complies with. For example, a new revision of this specification may add an operation to the Trigger Manager interface, incrementing the minor version. A client can detect support of the new operation by checking the interface version for the corresponding minor version, as defined in the above RULE in the specification revision which defines the new operation. Similarly, a Trigger Manager client implemented to use a major interface version of 1 can check the Version attribute to ensure the major version is 1 and therefore that the Trigger Manager still supports all requests the client may make. If the major version of a Trigger Manager is 2, a client that does not comprehend major version 2 should not call the Trigger Manager.

PERMISSION: A vendor key that is a child of the Trigger Managers category key MAY have two optional attributes, *Library*, whose type is String, and *Version*, whose type is Integer, to specify a vendor default Trigger Manager to be used for chassis created by the specified vendor, but which do not have a Trigger Manager model key specified.

RULE: A Trigger Manager vendor key SHALL NOT have either the Library attribute or the Version attribute described in the permission above unless it has both attributes.

RULE: The value of the Library attribute described in the above permission SHALL be the path to the library implementing the Trigger Manager for that vendor.

RULE: The value of the Version attribute described in the above permission SHALL be 0x00010000.

OBSERVATION: To provide backwards compatibility, any vendor's Trigger Manager may be selected as the default trigger manager for the system. The selection mechanism is described in *PXI-2: PXI Software Specification*.

An example of the Services Tree is shown below. The *Chassis*, *System Modules*, and *Peripheral Modules* portions of the Services Tree, as described in *PXI-6: PXI Express Software Specification*, are shown for context only. [Bracketed] lines indicate keys, with attributes indicated as <*name of attribute*> = <*value of attribute*>.

```
[Services]
    [Chassis]
    [Peripheral Modules]
    [System Modules]
    [Trigger Managers]
        [Vendor A]
            Library = C:\Vendor A\TrigMan.dll
            Version = 0x00010000
            [PXI-Chassis-A]
                Library = C:\Vendor A\ChassisATrigMan.dll
                Version = 0x00010000
        [Vendor B]
            [PXI-Chassis-B]
                Library = C:\Vendor B\TrigMan.dll
                Version = 0x00010000
            [PXI-Chassis-C]
                Library = C:\Vendor B\TrigMan.dll
                Version = 0x00010000
```

In the example, Vendor A has provided a chassis-specific Trigger Manager for PXI-Chassis-A, as well as a generic Trigger Manager for Vendor A's chassis other than PXI-Chassis-A. Vendor B has used a single Trigger Manager to implement trigger management capabilities for PXI-Chassis-B and PXI-Chassis-C, but specified no vendor default.

This Page Intentionally Left Blank

3. Software Frameworks and Requirements

This section discusses the framework-specific details of a PXI or PXI Express Trigger Manager. It gives an overview of the software requirements for components, along with framework-specific definitions and bindings for the software library described in previous sections of this specification.

3.1 Overview

The *PXI-2: PXI Software Specification* and *PXI-6: PXI Express Software Specification* describe the software requirements for components in a PXI and PXI Express system and the supported frameworks for software. The *PXI-6: PXI Express Software Specification* also describes binding and linkage protocols for specific software frameworks. This specification builds on those definitions by defining the binding and linkage protocols for the Trigger Manager in each software framework.

3.2 32-bit Windows System Framework

3.2.1 Introduction

In addition to the requirements in *PXI-2: PXI Software Specification* and *PXI-6: PXI Express Software Specification*, this specification describes additional requirements for software components relating to a Trigger Manager on 32-bit Windows operating systems.

3.2.2 Trigger Manager Software Bindings

RULE: A Trigger Manager SHALL be implemented as a 32-bit Windows DLL, with each operation corresponding to an exported symbol of the DLL.

OBSERVATION: Multiple processes can load and call any Trigger Manager simultaneously. Trigger Managers should take this into account in the implementation.

RULE: A Trigger Manager DLL SHALL export all symbols by name.

RULE: A Trigger Manager DLL SHALL use stdcall as the calling convention for all entry points.

RULE: A Trigger Manager DLL SHALL implement optional parameters by accepting a NULL pointer from the user for the optional parameter.

RULE: A Trigger Manager DLL SHALL use the following C data types to represent the data types given in the function definitions.

Operation Data Type	Return Type	Input Parameter Type	Output Parameter Type	Notes
Integer	N/A	int32_t	int32_t *	
Integer […]	N/A	const int32_t*	N/A	Pointer to standard C array, 0-indexed
Status	int32_t	N/A	N/A	

Operation Data Type	Return Type	Input Parameter Type	Output Parameter Type	Notes
Session	N/A	uintptr_t	uintptr_t*	
String	N/A	const char*, pointing to a null terminated ASCII string of 256 characters or less	char*, pointing to a caller-allocated buffer of 256 ASCII characters, must be NULL teminated on return	

3.2.3 Services Tree Implementation

The Services Tree is implemented in the Windows registry, as described in this section.

RULE: The root of the services tree SHALL be located in the 32-bit Windows registry at the following key:

```
HKEY_LOCAL_MACHINE\SOFTWARE\PXISA\Services
```

RULE: Keys and values in the 32-bit services tree SHALL identify 32-bit Trigger Manager components.

RULE: The name of a key in the Services Tree SHALL be the key name of that key in the Windows registry.

RULE: An attribute SHALL be implemented as a value in the Windows registry.

RULE: The types of the Service Tree key attributes SHALL be implemented using the following registry types.

Services Tree Type	Windows Registry Type
Integer	DWORD (REG_DWORD)
String	String (REG_SZ)

3.3 32-Bit Linux System Framework

3.3.1 Introduction

In addition to the requirements in *PXI-2: PXI Software Specification* and *PXI-6: PXI Express Software Specification*, this specification describes additional requirements for software components relating to a Trigger Manager on 32-bit Linux operating systems.

3.3.2 Trigger Manager Software Bindings

RULE: A Trigger Manager in the 32-bit Linux System Framework SHALL be implemented with the same 32-bit Linux Framework software bindings described in *PXI-6: PXI Express Software Specification*.

RULE: A Trigger Manager SHALL implement optional parameters by accepting a NULL pointer from the user for the optional parameter.

3.3.3 Service Tree Implementation

RULE: The Services Tree in the 32-bit Linux System Framework SHALL be implemented as described in *PXI-6: PXI Express Software Specification*.

3.3.4 Security of PXI Files and Interfaces

This section defines bindings to allow enforcement of a security policy on PXISA files and interfaces, in addition to those described in *PXI-2: PXI Software Specification*. The reader should refer to that specification for additional relevant text and details on the permissions notation used here.

RULE: Installation software for a Trigger Manager SHALL set the ownership and permissions of the Trigger Manager to be at least as permissive as pxisa:pxisa:550.

OBSERVATION: Some Trigger Manager implementations may involve the use of a daemon, files, or other entities that can have permissions of their own. The above RULE is intended to apply solely to the shared object that exposes Trigger Manager operations described in this specification. The permissions of other vendor-specific files, processes, or other resources implementing a Trigger Manager, while relevant to security, are outside the scope of this specification.

OBSERVATION: The permissions of 550 prevent users outside the pxisa group from tampering with route and reservation state. Vendors should consider the implications of exposing this functionality to other users before relaxing the permissions on the Trigger Manager in their implementation.

OBSERVATION: Restricting access to the Trigger Manager shared library alone will not completely prevent access to Trigger Manager functionality if the interfaces used by the Trigger Manager are exposed to all callers. Complete protection of such functionality requires that permissions be enforced down to the user/kernel boundary, or to some other boundary beyond which arbitrary users have no access. However, this may not always be practical; for example, an implementation may be built on top of an open source driver, which has many use cases unrelated to PXI, and therefore cannot be restricted.

RULE: Underlying mechanisms used in the implementation of a Trigger Manager SHALL have sufficiently permissive security implementation such that all callers which are members of the pxisa group will be capable of executing all of the Trigger Manager's operations.

3.4 64-Bit Windows System Framework

3.4.1 Introduction

In addition to the requirements in *PXI-2: PXI Software Specification* and *PXI-6: PXI Express Software Specification*, this specification describes additional requirements for software components relating to a Trigger Manager on 64-bit Windows operating systems.

3.4.2 Trigger Manager Software Bindings

RULE: A Trigger Manager SHALL be implemented as a 64-bit Windows DLL, with each operation corresponding to an exported symbol of the DLL.

RULE: A Trigger Manager implementation for 64-bit Windows operating systems SHALL implement both the 64-bit Trigger Manager DLL, as described in this section, and the 32-bit Trigger Manager DLL, as described in Section 3.2.2, *Trigger Manager Software Bindings*.

OBSERVATION: Multiple processes can load and call any Trigger Manager simultaneously. Trigger Managers should take this into account in the implementation.

OBSERVATION: The PXI Drivers defined in *PXI-6: PXI Express Software Specification* is called, in most cases, by the PXI Resource Manager, which is a 32-bit component. A 64-bit implementation of PXI Drivers is therefore not a requirement. A Trigger Manager is different; because its API is intended for use by higher level client applications, it must be implemented as both a 32-bit DLL and a 64-bit DLL to allow access to trigger resources by all applications.

RULE: A Trigger Manager DLL SHALL export all symbols by name.

RULE: A 64-bit Trigger Manager DLL SHALL use the same C data types as the 32-bit System Framework, as described in section 3.2.2.

RULE: A Trigger Manager DLL SHALL implement optional parameters by accepting a NULL pointer from the user for the optional parameter.

RULE: 32-bit and 64-bit Trigger Manager DLLs for the same chassis SHALL respect and enforce each other's state when making and clearing reservations and routes.

3.4.3 Services Tree Implementation

The Services Tree is implemented in the Windows registry, as described in this section.

RULE: The 32-bit services tree SHALL be implemented as described in section 3.2.3.

RULE: The root of the 64-bit services tree SHALL be located in the 64-bit Windows registry at the following key:

```
HKEY_LOCAL_MACHINE\SOFTWARE\PXISA\Services
```

RULE: Keys and values in the 64-bit services tree SHALL identify 64-bit Trigger Manager components.

RULE: The name of a key in the either Services Tree SHALL be the key name of that key in the Windows registry.

RULE: An attribute SHALL be implemented as a value in the Windows registry.

RULE: The types of the Service Tree key attributes SHALL be implemented using the following registry types.

Services Tree Type	Windows Registry Type
Integer	DWORD (REG_DWORD)
String	String (REG_SZ)

3.5 64-Bit Linux System Framework

3.5.1 Introduction

In addition to the requirements in *PXI-2: PXI Software Specification* and *PXI-6: PXI Express Software Specification*, this specification describes additional requirements for software components relating to a Trigger Manager on 64-bit Linux operating systems.

3.5.2 Trigger Manager Software Bindings

RULE: A Trigger Manager in the 64-bit Linux System Framework SHALL be implemented with the same 64-bit Linux Framework software bindings described in *PXI-6: PXI Express Software Specification*.

RULE: A Trigger Manager SHALL implement optional parameters by accepting a NULL pointer from the user for the optional parameter.

3.5.3 Service Tree Implementation

RULE: The Services Tree in the 64-bit Linux System Framework SHALL be implemented as described in *PXI-6: PXI Express Software Specification*.

3.5.4 Security of PXI Files and Interfaces

RULE: The Security of PXI Files and Interfaces SHALL be as defined for the 32-bit Linux framework in section 3.3.4.

This Page Intentionally Left Blank

Appendix A: Trigger Manager API Example Code

This section provides C language examples of common use cases that clients of the Trigger Manager API would use. Note that, unlike production code, these examples do not handle error conditions.

Setting and Clearing a Reservation

Shows how to set and clear the reservation of line 2 on bus 3 of chassis 4.

```
tPXISA_Status status;

// Open a session to the desired chassis.
tPXISA_Session session;
status = PXISA_ChassisTrig_OpenChassis (
   4,
   "PXISA_CLIENT_2",
   &session
   );

// Make the reservation
status = PXISA_ChassisTrig_SetReservation (
   session,
   3, // bus
   2, // line,
   kPXISA_Trig_Reserved
   );

////////////////////////////////////////////////
// Client-specific code that uses the reserved
// line goes here.
////////////////////////////////////////////////

// Unreserve the line
status = PXISA_ChassisTrig_SetReservation (
   session,
   3, // bus
   2, // line
   kPXISA_Trig_NotReserved
   );

PXISA_ChassisTrig_CloseChassis (
   session
   );
```

Reserving Multiple Lines Simultaneously

Sets a reservation on line 1 of buses 1, 2, and 3 simultaneously.

```
tPXISA_Status status;

// Prepare bus and line lists
tPXISA_Integer buses[3] = {1,2,3};
tPXISA_Integer lines[3] = {1,1,1};
 tPXISA_Integer failureIndex = 0;
```

```
// Open a session to the desired chassis.
tPXISA_Session session;
status = PXISA_ChassisTrig_OpenChassis (
   3,
   "PXISA_CLIENT_1",
   &session
   );

// Reserve the lines
status = PXISA_ChassisTrig_SetReservationMultiple (
   session,
   3, // reserving three lines (line 1, on three bus segments)
   buses, // buses to reserve
   lines, // lines to reserve
   &failureIndex
   );


PXISA_ChassisTrig_CloseChassis (
   session
   );
```

Setting a Route

Shows how to set a route on chassis 3 from bus 1, line 5 to bus 2, line 7.

```
tPXISA_Status status;

// Open a session to the desired chassis.
tPXISA_Session session;
status = PXISA_ChassisTrig_OpenChassis (
   3,
   "PXISA_CLIENT_1",
   &session
   );

// Reserve the destination line for the route.
status = PXISA_ChassisTrig_SetReservation (
   session,
   2, // bus
   7, // line
   kPXISA_Trig_Reserved
   );

// Set the desired route
status = PXISA_ChassisTrig_SetRoute (
   session,
   1, // source bus
   5, // source line
   2, // destination bus
   7  // destination line
   );
```

```
PXISA_ChassisTrig_CloseChassis (
   session
   );
```

Clearing a Route

Shows how to clear the route set above.

```
tPXISA_Status status;

// Open a session to the desired chassis.
tPXISA_Session session;
status = PXISA_ChassisTrig_OpenChassis (
   3,
   "PXISA_CLIENT_1",
   &session
   );

// Clear the desired route
status = PXISA_ChassisTrig_ClearRoute (
   session,
   2, // destination bus
   7  // destination line
   );

// Unreserve the destination line for the route.
status = PXISA_ChassisTrig_SetReservation (
   session,
   2, // bus
   7, // line
   kPXISA_Trig_NotReserved
   );

PXISA_ChassisTrig_CloseChassis (
   session
   );
```

Printing Line State

Shows how to print the line state of line 3 of bus 1 on chassis 6.

```
tPXISA_Status status;

// Open a session to the desired chassis.
tPXISA_Session session;
status = PXISA_ChassisTrig_OpenChassis (
   6,
   "PXISA_CLIENT_3",
   &session
   );

tPXISA_Integer reserveState, routeSrcBus, routeSrcLine;
tPXISA_Char owner[kPXISA_StringLength];

// Get the desired line state.
status = PXISA_ChassisTrig_GetLineInformation (
```

```
      session,
      1, // bus
      3, // line,
      &reserveState,
      &routeSrcBus,
      &routeSrcLine,
      owner
      );

   // We're done with the chassis - close the session.
   PXISA_ChassisTrig_CloseChassis (
      session
      );

   if (reserveState == kPXISA_Trig_Reserved)
   {
      printf("Bus 1 line 3 is reserved by client %s\n", owner);
   }
   else if (reserveState == kPXISA_Trig_Routed)
   {
      printf("Bus 1 line 3 is routed by client %s\n", owner);
      printf("  Source bus : %d\n", routeSrcBus);
      printf("  Source line: %d\n", routeSrcLine);
   }
   else if (reserveState == kPXISA_Trig_NotReserved)
   {
      printf("Bus 1 line 3 is not reserved\n");
   }
```

Appendix B: 32-Bit and 64-Bit Windows System Framework Files and Client Example

PXITriggerManager.h

```
#if !defined (__pxitriggermanager_h__)
#define __pxitriggermanager_h__

////////////////////////////////////////////////////////////////////////////
//
// Title    : PXITriggerManager.h
// Date     : 10/7/2011
// Purpose  : Definitions for using the API of the PXI Trigger Manager,
//   compliant with version 1.0 of the PXI and PXI Express Trigger Management
//   Specification. Note that this header requires the use of C99 data types.
//   The client of this file is required to ensure that these types are defined
//   before including this file, generally by including stdint.h beforehand.
//
////////////////////////////////////////////////////////////////////////////

#if defined(_WIN32) && !defined(_WIN64)
#define PXISA_FUNC __stdcall
#else
#define PXISA_FUNC
#endif

////////////////////////////////////////////////////////////////////////////
//
// Definition of types
//
////////////////////////////////////////////////////////////////////////////

typedef int32_t                     tPXISA_Status;
typedef int32_t                     tPXISA_Integer;
typedef tPXISA_Integer *            tPXISA_PInteger;
typedef const tPXISA_Integer *      tPXISA_PIntegerConstant;
typedef char                        tPXISA_Char;
typedef const tPXISA_Char *         tPXISA_StringConstant;
typedef tPXISA_Char *               tPXISA_String;
typedef uintptr_t                   tPXISA_Session;
typedef tPXISA_Session *            tPXISA_PSession;

////////////////////////////////////////////////////////////////////////////
//
// Definition of status codes
//
////////////////////////////////////////////////////////////////////////////
enum ePXISA_Trig_Status
{
   // Status range specifiers
   // Set to avoid conflict with PXIExpress.h from PXI-6
   kPXISA_ErrorCodeStart   = -2,
```

```
    kPXISA_WarningCodeStart =  2,

    // Errors

    kPXISA_ErrorUnsupported             = (kPXISA_ErrorCodeStart-0),
    kPXISA_ErrorInvalidParameter        = (kPXISA_ErrorCodeStart-1),
    kPXISA_ErrorLineNotReserved         = (kPXISA_ErrorCodeStart-2),
    kPXISA_ErrorLineAlreadyReserved     = (kPXISA_ErrorCodeStart-3)
    kPXISA_ErrorConflictingRoute        = (kPXISA_ErrorCodeStart-4),
    kPXISA_ErrorInvalidClient           = (kPXISA_ErrorCodeStart-5),
    kPXISA_ErrorDisconnected            = (kPXISA_ErrorCodeStart-6)

    // Warnings

};

////////////////////////////////////////////////////////////////////////////
//
// Trigger Manager Interface
//
////////////////////////////////////////////////////////////////////////////

#define kPXISA_ChassisTrig_OpenChassis_String "PXISA_ChassisTrig_OpenChassis"

typedef tPXISA_Status (PXISA_FUNC * tPXISA_ChassisTrig_OpenChassis) (
    tPXISA_Integer        chassisNum,
    tPXISA_StringConstant clientLabel,
    tPXISA_PSession       session
    );

#define kPXISA_ChassisTrig_CloseChassis_String "PXISA_ChassisTrig_CloseChassis"

typedef void (PXISA_FUNC * tPXISA_ChassisTrig_CloseChassis) (
    tPXISA_Session        session
    );

// Possible values for reservation state
enum ePXISA_TriggerLineState
{
    kPXISA_Trig_NotReserved      = 0,
    kPXISA_Trig_Reserved         = 1,
    kPXISA_Trig_Routed           = 2
};

#define kPXISA_ChassisTrig_SetReservation_String
"PXISA_ChassisTrig_SetReservation"

typedef tPXISA_Status (PXISA_FUNC * tPXISA_ChassisTrig_SetReservation) (
    tPXISA_Session        session,
    tPXISA_Integer        bus,
    tPXISA_Integer        line,
    tPXISA_Integer        reserve
    );
```

```
#define kPXISA_ChassisTrig_SetReservationMultiple_String
"PXISA_ChassisTrig_SetReservationMultiple"

typedef tPXISA_Status (PXISA_FUNC * tPXISA_ChassisTrig_SetReservationMultiple) (
   tPXISA_Session          session,
   tPXISA_Integer          numElements,
   tPXISA_PIntegerConstant buses,
   tPXISA_PIntegerConstant lines,
   tPXISA_PInteger         indexOfFailure
   );

#define kPXISA_ChassisTrig_SetRoute_String "PXISA_ChassisTrig_SetRoute"

typedef tPXISA_Status (PXISA_FUNC * tPXISA_ChassisTrig_SetRoute) (
   tPXISA_Session        session,
   tPXISA_Integer        sourceBus,
   tPXISA_Integer        sourceLine,
   tPXISA_Integer        destBus,
   tPXISA_Integer        destLine
   );

#define kPXISA_ChassisTrig_ClearRoute_String "PXISA_ChassisTrig_ClearRoute"

typedef tPXISA_Status (PXISA_FUNC * tPXISA_ChassisTrig_ClearRoute) (
   tPXISA_Session        session,
   tPXISA_Integer        destBus,
   tPXISA_Integer        destLine
   );

#define kPXISA_ChassisTrig_GetLineInformation_String
"PXISA_ChassisTrig_GetLineInformation"

typedef tPXISA_Status (PXISA_FUNC * tPXISA_ChassisTrig_GetLineInformation) (
   tPXISA_Session        session,
   tPXISA_Integer        bus,
   tPXISA_Integer        line,
   tPXISA_PInteger       reserveState,
   tPXISA_PInteger       routeSrcBus,
   tPXISA_PInteger       routeSrcLine,
   tPXISA_String         owner
   );

#define kPXISA_ChassisTrig_ClearAllRoutesAndReservations_String
"PXISA_ChassisTrig_ClearAllRoutesAndReservations"

typedef tPXISA_Status (PXISA_FUNC *
tPXISA_ChassisTrig_ClearAllRoutesAndReservations) (
   tPXISA_Session        session
   );

#endif // #if !defined (__pxitriggermanager_h__)
```

PXITriggerManager.def

```
; Module definition file for a PXI Trigger Manager
EXPORTS
   PXISA_ChassisTrig_OpenChassis
   PXISA_ChassisTrig_CloseChassis
   PXISA_ChassisTrig_SetReservation
   PXISA_ChassisTrig_SetReservationMultiple
   PXISA_ChassisTrig_SetRoute
      PXISA_ChassisTrig_ClearRoute
   PXISA_ChassisTrig_GetLineInformation
   PXISA_ChassisTrig_ClearAllRoutesAndReservations
```

Client Example

The series of steps below describes the sequence of steps that are executed by a client in order to locate and use the Trigger Manager for chassis 2, model *PXI Chassis*, manufactured by vendor *PXI Vendor* on a system running Microsoft Windows. Note that a thorough understanding of these steps relies on an understanding of mechanisms described throughout this document, as well as Trigger Manager information from the system description file(s), the population of which is described in *PXI-2: PXI Software Specification* and *PXI-6: PXI Express Software Specification*.

1. The client requires access to trigger resources in chassis 2.
2. The client reads the `pxisys.ini` file and/or `pxiesys.ini` file searching for the description of chassis 2. Refer to the PXI-2 and PXI-6 specifications for descriptions of the `pxisys.ini` and `pxiesys.ini` file formats, respectively.

   ```
   [Chassis2]
   ...
   TriggerManager="PXI Vendor\PXI Chassis"
   ...
   ```

3. The client reads the *TriggerManager* attribute from the *[Chassis2]* section of the system description file.
4. The client appends the attribute value from the system description file to the path to the *Trigger Managers* section of the Services Tree, resulting in:

   <*path to services tree root*>\Trigger Managers\PXI Vendor\PXI Chassis
5. The client reads the relevant section of the Services Tree to find that the appropriate Trigger Manager for chassis 2 is `C:\Program Files\PXI Vendor\TriggerManager1.dll`.

   [...]
   [Services]
   [Trigger Managers]
   [PXI Vendor]
   [PXI Chassis]
   Library = C:\Program Files\PXI Vendor\TriggerManager1.dll
   Version = 0x00010000
   [PXI Some Other Chassis]
   Library = C:\Program Files\PXI Vendor\TriggerManager2.dll
   Version = 0x00010000
   [Some Other PXI Vendor]
   Library = C:\Program Files\Some Other PXI Vendor\DefaultTriggerManager.dll
   Version = 0x00010000

6. The client loads `TriggerManager1.dll` and opens a session to chassis 2 using some client label.
7. The client accesses the Trigger Manager as necessary. The client can consult the system description file(s) for information on the trigger line routing and reservation capabilities provided by the chassis, but the Trigger Manager will also enforce the hardware limitations by returning errors when appropriate.
8. The client may close its session to the Trigger Manager, or it may continue to leave it open if other reservations or routes may need to be changed.
9. The client performs operations outside of the Trigger Manager which drive the trigger lines that have been reserved.
10. If the client previously closed its session, it reopens the session with the same client label used previously.
11. When the client is finished with the trigger resources, it clears its reservations and routes and closes the session.