Systems Alliance

VPP-3.2: Instrument Driver Functional Body Specification

Revision 5.1

April 14, 2008

VPP-3.2 Revision History

This section is an overview of the revision history of the VPP-3.2 specification.

Revision 1.0, July 15, 1994

This edition reflects a non-technical revision for style and format issues.

Revision 1.1, August 17, 1994

This edition reflects edits to technical omissions and inconsistencies between VPP documents.

Revision 2.0, November 28, 1994

This edition reflects technical changes made at the November 3, 1994 Technical Working Group and subsequent edits made during review.

Revision 3.0, February 21, 1995

This edition reflects the addition of new completion and error codes.

Revision 4.0, February 2, 1996

This edition is the result of a reorganization of the entire VPP-3.*X* series of specifications. The goal of the reorganization was to improve the documentation of instrument driver requirements for all frameworks. The title of the specification has been changed to VPP-3.2, *Instrument Driver Functional Body Specification*.

Revision 5.0, December 4, 1998

This edition creates Appendix A that specifies optional auto-connect functions for instrument drivers and updates the information regarding contacting the alliance. References to the VPP-5 Component Knowledge Base specification, which was obsoleted by the alliance, were removed.

Revision 5.1, February 14, 2008

Updated the introduction to reflect the IVI Foundation organization changes. Replaced Notice with text used by IVI Foundation specifications..

Revision 5.1, April 14, 2008

Editorial change to update the IVI Foundation contact information in the Important Information section to remove obsolete address information and refer only to the IVI Foundation web site.

NOTICE

VPP-3.2: *Instrument Driver Functional Body Specification* is authored by the IVI Foundation member companies. For a vendor membership roster list, please visit the IVI Foundation web site at `www.ivifoundation.org`.

The IVI Foundation wants to receive your comments on this specification. You can contact the Foundation through the web site at `www.ivifoundation.org`.

Warranty

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

Trademarks

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

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

Contents

Section 1
Introduction to the VXI*plug&play* Systems Alliance and the IVI Foundation

---

The VXI*plug&play* Systems Alliance was founded by members who shared a common commitment to end-user success with open, multivendor VXI systems. The alliance accomplished major improvements in ease of use by endorsing and implementing common standards and practices in both hardware and software, beyond the scope of the VXIbus specifications. The alliance used both formal and de facto standards to define complete system frameworks. These standard frameworks gave end-users "plug & play" interoperability at both the hardware and system software level.

The IVI Foundation is an organization whose members share a common commitment to test system developer success through open, powerful, instrument control technology. The IVI Foundation's primary purpose is to develop and promote specifications for programming test instruments that simplify interchangeability, provide better performance, and reduce the cost of program development and maintenance.

In 2002, the VXI*plug&play* Systems Alliance voted to become part of the IVI Foundation. In 2003, the VXI*plug&play* Systems Alliance formally merged into the IVI Foundation. The IVI Foundation has assumed control of the VXI*plug&play* specifications, and all ongoing work will be accomplished as part of the IVI Foundation.

All references to VXI*plug&play* Systems Alliance within this document, except contact information, were maintained to preserve the context of the original document.

Section 2
Overview of the Instrument Driver Functional Body Specification

2.1 Introduction

This section introduces the *Instrument Driver Functional Body Specification*, which was written by the VXI*plug&play* Systems Alliance. The Instrument Driver Technical Working Group both wrote the specification and performed the technical work that the specification discusses.

This section summarizes the *Instrument Driver Functional Body Specification* itself and contains general information that the reader may need in order to understand, interpret, and implement aspects of this specification. These aspects include the following:

- The objectives of the specification
- The audience of the specification
- The scope and organization of the specification
- The application of the specification
- The definitions of terms and acronyms
- References
- Conventions

2.2 Objectives of This Specification

The *Instrument Driver Functional Body Specification* provides a general overview of the development of multi-vendor VXI*plug&play* instrument drivers. This specification describes the instrument driver functional body and defines how the required instrument driver functions are implemented.

2.3 Audience for This Specification

This specification has two audiences. The first audience consists of instrument driver developers—either instrument vendors, system integrators, or end users—who want to implement instrument driver software that is compliant with this specification. The second audience consists of instrumentation end users and application programmers who want to implement applications that use instrument drivers compliant with this specification.

2.4 Scope and Organization of This Specification

This specification is organized in sections. Each section discusses a particular aspect of the VXI*plug&play* Systems Alliance standard for instrument drivers.

Section 1 explains the VXI*plug&play* Systems Alliance and its relation to the IVI Foundation.

Section 2 summarizes this specification and discusses its objectives, scope, organization, application, references, term definitions, acronyms, and conventions.

Section 3 summarizes the operation of the required VXI*plug&play* instrument driver functions.

Section 4 discusses the instrument driver prefix defines how it applies to the naming conventions for instrument driver functions and filenames.

Section 5 defines the required instrument driver functions for C language required instrument driver functions.

Section 6 defines the required instrument driver functions for G language required instrument driver functions.

2.5 Application of This Specification

This specification is intended to be used by developers of VXI*plug&play* instrument drivers. It is also useful as a reference for end users of VXI*plug&play* instrument drivers.

This specification is intended to be used in conjunction with the *Instrument Drivers Architecture and Design Specification* (VPP-3.1), the *Instrument Driver Interactive Developer Interface Specification* (VPP-3.3), the *Instrument Driver Programmatic Developer Interface Specification* (VPP-3.4) and the *VISA Specifications* (VPP-4.*x*). VXI*plug&play* instrument drivers developed in accordance with these specifications can be used in a wide variety of higher level software environments, as described in the VXI*plug&play System Frameworks Specification* (VPP-2).

2.6 References

Several other VXI*plug&play* Systems Alliance documents and specifications are related to this specification. These other related documents are the following:

- VPP-2 *System Frameworks Specification*
- VPP-3.1 *Instrument Drivers Architecture and Design Specification*
- VPP-3.3 *Instrument Driver Interactive Developer Interface Specification*
- VPP-3.4 *Instrument Driver Programmatic Developer Interface Specification*
- VPP-4.x *Virtual Instrument Software Architecture Specifications*
- VPP-6 *Installation and Packaging Specification*
- VPP-7 *Soft Front Panel Specification*
- VPP-9 *Instrument Vendor Abbreviations*

2.7 Definitions of Terms and Acronyms

• ADE	Application Development Environment
• LabWindows/CVI	C-based ADE
• LabVIEW	Graphical-programming ADE
• Instrument Driver	Library of functions for controlling a specific instrument
• Template or Required Function	Instrument Driver function required to be implemented in all VXI*plug&play* instrument drivers
• Application Function	A high-level, test-oriented, instrument driver function. It is typically developed from the instrument driver subsystem functions.
• VISA	Virtual Instrument Software Architecture
• VI	LabVIEW program or Virtual Instrument
• LLB	LabVIEW VI library

- DLL Dynamic Link Library

2.8 Conventions

The following headings distinguish paragraphs throughout this specification. These headings give special meaning to the paragraphs.

Rules must be followed to ensure compatibility with the System Framework. A rule is characterized by the words **SHALL** or **SHALL NOT** in bold upper case characters. **SHALL** or **SHALL NOT** are not used in this manner for any other purpose.

Recommendations contain advice to implementors. This advice affects the usability of the final device. Recommendations are included in this specification to draw attention to particular characteristics that the authors believe to be important to end-user success.

Permissions authorize specific implementations or uses of system components. A permission is characterized by the word **MAY** in bold upper case characters. These permissions are granted to ensure that specific system framework components are well defined and can be tested for compatibility and interoperability.

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

A Note on the text of the specification: Any text that appears without heading should be considered a description of the standard and how the architecture was intended to operate. The purpose of this text is to give the reader a deeper understanding of the intentions of the specification, including the underlying model and specific required features. The implementor of this standard should ensure that a particular implementation does not conflict with the text of the standard.

Section 3
Required Instrument Driver Functions

3.1 Introduction

This section defines which functions are required in an instrument driver and defines their operation. This section describes the required instrument driver functions in non-framework specific terms. Prototypes for implementing required functions for C and G language instrument drivers are described in Sections 5 and 6.

RULE 3.1

A VXI*plug&play* instrument driver **SHALL** implement the required instrument driver functions: Initialize, Close, Reset, Self-Test, Error Query, and Revision Query.

RULE 3.2

All VXI*plug&play* C language instrument drivers **SHALL** implement the Error Message function.

PERMISSION 3.1

G language VXI*plug&play* instrument drivers **MAY** implement the Error Message function.

OBSERVATION 3.1

G language VXI*plug&play* instrument drivers make use other error handling techniques within the ADE to retrieve error information.

RULE 3.3

The operation of VXI*plug&play* required instrument driver functions **SHALL** be as defined in Sections 3.2 through 3.8.

3.2 Initialize

Purpose

To establish communication with the instrument.

Parameters

Inputs	Description	Base Type
rsrcName	Instrument Description	ViRsrc
id_query	If (VI_TRUE) perform In-System Verification. If (VI_FALSE) do not perform In-System Verification.	ViBoolean
reset_instr	If (VI_TRUE) perform Reset operation. If (VI_FALSE) do not perform Reset operation.	ViBoolean

Outputs	Description	Base Type
vi	Instrument handle	ViSession

Return Values

Type ViStatus — Operational return status. Contains either a completion code or an error code. Instrument driver specific codes that may be returned in addition to the VISA error codes defined in VPP-4.3 and vendor specified codes, are as follows.

Completion Codes	Description
VI_SUCCESS	Initialization successful
VI_WARN_NSUP_ID_QUERY	Identification query not supported
VI_WARN_NSUP_RESET	Reset not supported

Error Codes	Description
VI_ERROR_FAIL_ID_QUERY	Instrument identification query failed

OBSERVATION 3.2

Establishing communication with the instrument may require configuring communication options such as turning headers on or off, using long or short form queries, and so on, within the instrument as well as configuring the I/O interface between the instrument and the computer.

RULE 3.4

IF the user sets the reset_instr parameter to VI_TRUE, **THEN** the function **SHALL** place the instrument in a pre-defined reset state.

When a user sets the `reset_instr` parameter to `VI_TRUE`, the function should place the instrument in the same state as that of the `prefix_reset` function.

RULE 3.5

The initialize function **SHALL** allow the user to select whether to verify the identity of the instrument. Verifying the identity can be accomplished by checking the manufacturer ID and model number in the instrument's VXI register set, by using the *IDN query for IEEE 488.2 compatible instruments, or by other means.

RULE 3.6

IF the user sets the `id_query` parameter to `VI_TRUE` on an instrument that does not perform that functionality, **THEN** the function **SHALL** return the warning `VI_WARN_NSUP_ID_QUERY`.

RULE 3.7

IF the user sets the `reset_instr` parameter to `VI_TRUE` on an instrument that cannot be programmatically reset to a known state, **THEN** the function **SHALL** return the warning `VI_WARN_NSUP_RESET`.

RULE 3.8

IF the initialize function encounters an error, **THEN** the value of the `vi` output parameter **SHALL** be `VI_NULL` and any valid sessions obtained from `viOpen` **SHALL** be closed.

3.3 Reset

Purpose

Places the instrument in a default state.

Parameters

Inputs	Description	Base Type
`vi`	Instrument handle	`ViSession`

Return Values

Type `ViStatus` Operational return status. Contains either a completion code or an error code. Instrument driver specific codes that may be returned in addition to the VISA error codes defined in VPP-4.3 and vendor specified codes, are as follows.

Completion Codes	Description
`VI_SUCCESS`	Reset operation successful
`VI_WARN_NSUP_RESET`	Reset not supported

RULE 3.9

IF your instrument cannot be reset programmatically, **THEN** the function **SHALL** return the warning `VI_WARN_NSUP_RESET`.

RECOMMENDATION 3.1

The default state in which the Reset function places the instrument, as well as how long this operation will take, should be documented in the help information for the Reset function.

3.4 Self-Test

Purpose

This function causes the instrument to perform a self-test and returns the result of that self-test.

Parameters

Inputs	Description	Base Type
`vi`	Instrument handle	`ViSession`

Outputs	Description	Base Type
`test_result`	Numeric result from self-test operation 0 = no error (test passed)	`ViInt16`
`test_message`	Self-test status message	`ViString`

Return Values

Type `ViStatus` Operational return status. Contains either a completion code or an error code. Instrument driver specific codes that may be returned in addition to the VISA error codes defined in VPP-4.3 and vendor specified codes, are as follows.

Completion Codes	Description
`VI_SUCCESS`	Self-test operation successful
`VI_WARN_NSUP_SELF_TEST`	Self-test not supported

RULE 3.10

IF your instrument cannot perform a self-test, **THEN** the function **SHALL** return the warning `VI_WARN_NSUP_SELF_TEST`.

RULE 3.11

This function **SHALL** write a maximum of 256 characters including the null byte in the `test_message` output buffer.

RECOMMENDATION 3.2

The time to complete the Self-Test operation should be documented in the help information for the function.

3.5 Error Query

Purpose

This function queries the instrument and returns instrument-specific error information.

Parameters

Inputs	Description	Base Type
`vi`	Instrument handle	`ViSession`

Outputs	Description	Base Type
`error_code`	Instrument error code	`ViInt32`
`error_message`	Error message	`ViString`

Return Values

Type `ViStatus` Operational return status. Contains either a completion code or an error code. Instrument driver specific codes that may be returned in addition to the VISA error codes defined in VPP-4.3 and vendor specified codes, are as follows.

Completion Codes	Description
`VI_SUCCESS`	Error query successful
`VI_WARN_NSUP_ERROR_QUERY`	Error query not supported

RULE 3.12

IF your instrument cannot perform an error query, **THEN** the function **SHALL** return the warning `VI_WARN_NSUP_ERROR_QUERY`.

RULE 3.13

This function **SHALL** write a maximum of 256 characters including the null byte in the `error_message` output buffer.

3.6 Error Message

Purpose

This function translates the error return value from a VXI*plug&play* instrument driver function to a user-readable string.

Parameters

Inputs	Description	Base Type
`vi`	Instrument handle	`ViSession`
`status_code`	Instrument driver error code	`ViStatus`

Outputs	Description	Base Type
`message`	VISA or instrument driver Error message	`ViString`

Return Values

Type `ViStatus` Operational return status. Contains either a completion code or an error code. Instrument driver specific codes that may be returned in addition to the VISA error codes defined in VPP-4.3 and vendor specified codes, are as follows.

Completion Code	Description
`VI_SUCCESS`	Error message function successful.

RULE 3.14

The Error Message function **SHALL** accept a value of `VI_NULL` for the `vi` input parameter. **IF** the value `VI_NULL` is passed into the function as the parameter `vi`, **THEN** it **SHALL** be ignored; otherwise, the value of the `vi` parameter may be used by the function.

RULE 3.15

This function **SHALL** write a maximum of 256 characters including the null byte in the `message` output buffer.

RULE 3.16

IF the Error Message function cannot interpret the value in `status_code`, **THEN** it **SHALL** return `VI_WARN_UNKNOWN_STATUS`.

3.7 Revision Query

Purpose

Returns the revision of the instrument driver and the firmware revision of the instrument being used.

Parameters

Inputs	Description	Base Type
vi	Instrument handle	ViSession

Outputs	Description	Base Type
driver_rev	Instrument driver revision	ViString
instr_rev	Instrument firmware revision	ViString

Return Values

Type ViStatus — Operational return status. Contains either a completion code or an error code. Instrument driver specific codes that may be returned in addition to the VISA error codes defined in VPP-4.3 and vendor specified codes, are as follows.

Completion Codes	Description
VI_SUCCESS	Revision query successful
VI_WARN_NSUP_REV_QUERY	Instrument revision query not supported

RULE 3.17

IFthe instrument firmware revision is not supported by the instrument, **THEN** the Revision Query function **SHALL** return the literal string “Not Available” in the instrument revision output parameter, and the function **SHALL** return the warning VI_WARN_NSUP_REV_QUERY.

RULE 3.18

This function **SHALL** write a maximum of 256 characters including the null byte in the driver_rev output buffer.

RULE 3.19

This function **SHALL** write a maximum of 256 characters including the null byte in the instr_rev output buffer.

3.8 Close

Purpose

Terminates the software connection to the instrument and deallocates system resources associated with that instrument.

Parameters

Inputs	Description	Base Type
`vi`	Instrument handle	`ViSession`

Return Values

Type `ViStatus` Operational return status. Contains either a completion code or an error code. Instrument driver specific codes that may be returned in addition to the VISA error codes defined in VPP-4.3 and vendor specified codes, are as follows.

Completion Codes	Description
`VI_SUCCESS`	Close successful

Section 4
Instrument Driver Naming Conventions

4.1 Introduction

To guarantee that VXI*plug&play* instrument driver developers deliver instrument drivers with unique filenames, the VXI*plug&play* Systems Alliance has defined specifications for the naming of instrument driver filename prefixes. These prefixes are also used to uniquely name instrument driver functions within a given instrument driver.

4.2 Instrument Driver Prefixes

The prefixes are maintained by the VXI*plug&play* Systems Alliance. A developer who wants a new prefix must notify the alliance. Vendors do not need to join the VXI*plug&play* Systems Alliance to obtain a defined prefix.

RULE 4.1

All VXI*plug&play* instrument drivers **SHALL** have a PREFIX. This PREFIX **SHALL** begin with two characters based on the instrument vendor as defined in VPP-9, *Instrument Vendor Abbreviations* followed by characters that uniquely identify the module.

RULE 4.2

All required VXI*plug&play* instrument driver files **SHALL** have a filename that is the PREFIX plus file type extension, (EXAMPLE: PREFIX.c).

PERMISSION 4.1

VXI*plug&play* instrument drivers **MAY** have more than one file with the same extension. These additional files **MAY** have a filename that is not identical to the PREFIX of the instrument driver, but must begin with the two character abbreviation for the manufacturer.

RULE 4.3

All exported symbols from a VXI*plug&play* instrument driver **SHALL** begin with the PREFIX.

For example, the filename for the Tektronix VX4790 Arbitrary Waveform Generator could be `tkvx4790`. The prefix `tkvx4790` would also be used as the prefix for all functions found in the instrument driver.

OBSERVATION 4.1

PREFIX is used throughout the VXI*plug&play* specifications. Wherever the string PREFIX appears in formal syntax, the actual driver prefix defined in this section should be substituted.

Section 5
Bindings for C Language Required Instrument Driver Functions

5.1 Introduction

This section defines the ANSI C and Visual Basic bindings for C language required instrument driver functions.

5.2 ANSI C Bindings

This section defines the function prototypes for the ANSI C bindings for the required instrument driver functions from Section 3.

```
ViStatus _VI_FUNC PREFIX_init(ViRsrc rsrcName, ViBoolean id_query,
                              ViBoolean reset_instr, ViPSession vi);

ViStatus _VI_FUNC PREFIX_close(ViSession vi);

ViStatus _VI_FUNC PREFIX_reset(ViSession vi);

ViStatus _VI_FUNC PREFIX_self_test(ViSession vi, ViPInt16 test_result,
                                   ViChar test_message[]);

ViStatus _VI_FUNC PREFIX_revision_query(ViSession vi,
                                        ViChar driver_rev[],
                                        ViChar instr_rev[]);

ViStatus _VI_FUNC PREFIX_error_query(ViSession vi, ViPInt32 error_code,
                                     ViChar error_message[]);

ViStatus _VI_FUNC PREFIX_error_message(ViSession vi, ViStatus status_code,
                                       ViChar message[]);
```

RULE 5.1

All VXI*plug&play* C language instrument drivers **SHALL** implement the required driver functions with the ANSI C bindings defined in Section 5.2.

PERMISSION 5.1

A VXI*plug&play* C language instrument driver may use other names for parameters than those specified in Sections 5.2 and 5.3.

RULE 5.2

Every VXI*plug&play* instrument driver header file (`PREFIX.h`) **SHALL** contain function prototypes for the required driver functions as specified in Section 5.2.

Section 6
Bindings for G Language Required Instrument Driver Functions

6.1 Introduction

This section defines the required instrument driver function prototypes for the G language VXI*plug&play* instrument drivers.

The data types used in this section are defined in VPP-4.3.3: *VISA Implementation Specification for the G Language.*

RULE 6.1

All G language VXI*plug&play* instrument drivers **SHALL** implement the required instrument driver functions as defined in this section.

PERMISSION 6.1

The icon for the required operations may be different than those specified in this section.

6.2 Initialize

Wiring Diagram

Initialize.vi

Connector Pane

Inputs	
VISA resource name	abc
ID Query	TF
Reset	TF
error in	

Outputs	
VISA resource name out	U32
error out	

6.3 Reset

Wiring Diagram

Reset.vi

Connector Pane

Inputs	
VISA resource name	U32
error in	

Outputs	
VISA resource name out	U32
error out	

6.4 Self-Test

Wiring Diagram

Self-Test.vi

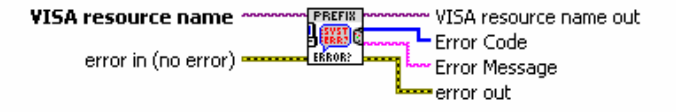
Connector Pane

Inputs	
VISA resource name	U32
error in	

Outputs	
VISA resource name out	U32
Self-Test Result Code	U32
Self-Test Result Message	abc
error out	

6.5 Error Query

Wiring Diagram



Error Query.vi

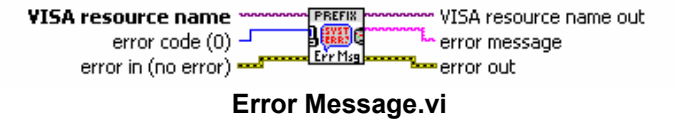
Connector Pane

Inputs	
VISA resource name	U32
error in	

Outputs	
VISA resource name out	U32
Error Code	U32
Error Message	abc
error out	

6.6 Error Message

Wiring Diagram



Error Message.vi

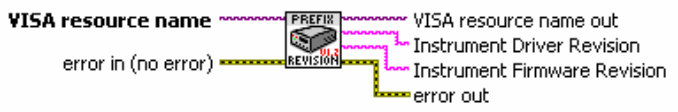
Connector Pane

Inputs	
VISA resource name	U32
error code	abc
error in	

Outputs	
VISA resource name out	U32
error message	abc
error out	

6.7 Revision Query

Wiring Diagram



Revision Query.vi

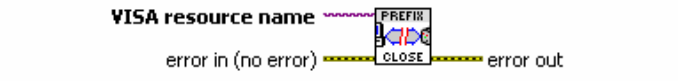
Connector Pane

Inputs	
VISA resource name	U32
error in	

Outputs	
VISA resource name out	U32
Instrument Driver Revision	abc
Instrument Firmware Revision	abc
error out	

6.8 Close

Wiring Diagram



Close.vi

Connector Pane

Inputs	
VISA resource name	U32
error in	

Outputs	
error out	

Appendix A
Completion and Error Codes

This appendix lists the completion and error codes. Tables A-1 and A-2 list the instrument driver completion and error codes, respectively.

Table A-1. Instrument Driver Completion Codes

Completion Codes	Description
VI_SUCCESS	No error: the call was successful
VI_WARN_NSUP_ID_QUERY	Identification query not supported
VI_WARN_NSUP_RESET	Reset not supported
VI_WARN_NSUP_SELF_TEST	Self-test not supported
VI_WARN_NSUP_ERROR_QUERY	Error query not supported
VI_WARN_NSUP_REV_QUERY	Revision query not supported

Table A-2. Instrument Driver Error Codes

Status	Description
VI_ERROR_FAIL_ID_QUERY	Instrument identification query failed
VI_ERROR_INV_RESPONSE	Error interpreting instrument response
VI_ERROR_PARAMETER1	Parameter 1 out of range
VI_ERROR_PARAMETER2	Parameter 2 out of range
VI_ERROR_PARAMETER3	Parameter 3 out of range
VI_ERROR_PARAMETER4	Parameter 4 out of range
VI_ERROR_PARAMETER5	Parameter 5 out of range
VI_ERROR_PARAMETER6	Parameter 6 out of range
VI_ERROR_PARAMETER7	Parameter 7 out of range
VI_ERROR_PARAMETER8	Parameter 8 out of range
0xBFFC0800 to 0xBFFC0FFF	Instrument-specific errors

Refer to VPP-4.*X*, *VISA Specifications*, for the VISA error codes.

Appendix B
Optional Instrument Driver Functions

B.1 Introduction

This section defines functions which are standardized but not required in an instrument driver . This section describes the standard instrument driver functions in non-framework specific terms. Prototypes for implementing standard functions for C and G language instrument drivers are described in later sub-sections.

RULE B.1
The operation of VXI*plug&play* optional autoConnect functions **SHALL** be as defined in Sections B.2.1 through B.2.4.

RULE B.2

The function names used in sections B.2.1 through B.2.4 are reserved and **SHALL** not be used for any other purpose in a VXI*plug&play* Instrument driver.

OBSERVATION B.1

If autoConnect functions are used there is no need to call prefix_init because the autoConnect function has done the initialize and returns a valid session handle.

RECOMMENDATION B.1

autoConnect functions should perform reset when implementing prefix_init functionality.

B.2 Optional Instrument Driver Functions

This section defines the optional instrument driver functions

B.2.1 autoConnectToFirst

Purpose

To establish communication with the instrument. This function searches the VXI system for a module supported by the driver. It establishes communication with the first module found. If more than one module of the same type exists in the system, autoConnectToSlot or autoConnectToLA may be used.

Parameters

Outputs	Description	Base Type
`vi`	Instrument handle	`ViSession`

Return Values

Type `ViStatus`	Operational return status. Contains either a completion code or an error code. Instrument driver specific codes that may be returned in addition to the VISA error codes defined in VPP-4.3 and vendor specified codes, are as follows.

Completion Codes	Description
`VI_SUCCESS`	Initialization successful
`VI_MORE_INST_PRESENT`	One valid session was returned. However, there are additional instruments in the system not accessed by this function.

RULE B.3

IF the autoConnectToFirst function encounters an error, **THEN** the value of the `vi` output parameter **SHALL** be `VI_NULL` and any valid sessions obtained from `viOpen` **SHALL** be closed.

OBSERVATION B.2

VISA may span more than a single VXI system. This function returns a session to the first one found.

RECOMMENDATION B.2

The first instrument should be first match using viFindRsc and viFindNext.

B.2.2 autoConnectToSlot

Purpose

To establish communication with the instrument. This function searches the VXI system for a module supported by the driver. It establishes communication with the module only if it is found in the specified slot. This function returns an array of instrument handles because multiple devices may exist in a single VXI slot.

Slot numbers are not guaranteed to be unique. In a multi-mainframe system where the slot numbers are not unique use autoConnectToLA.

Parameters

Inputs	Description	Base Type
`arrayLength`	Size of instrument handle array	`ViInt16`
`slot`	VXI slot number	`ViInt16`

Outputs	Description	Base Type
`instrArray`	Array of session handles for the selected slot	`ViSession []`
`numConnected`	The number of instrument handles in the array	`ViInt16`

Return Values

Type `ViStatus`	Operational return status. Contains either a completion code or an error code. Instrument driver specific codes that may be returned in addition to the VISA error codes defined in VPP-4.3 and vendor specified codes, are as follows.

Completion Codes	Description
`VI_SUCCESS`	Initialization successful
`VI_MORE_INST_PRESENT`	One or more valid sessions was returned. However, there are additional instruments in the system not accessed by this function.

RULE B.4

IF the autoConnectToSlot function encounters an error, **THEN** the value of the `numConnected` output parameter **SHALL** be zero and any valid sessions obtained from `viOpen` **SHALL** be closed.

RECOMMENDATION B.3

Any valid sessions obtained from viOpen that are subsequently closed should be equal to zero.

RECOMMENDATION B.4

Devices which do not have module ID capability should not implement this function.

B.2.3 autoConnectToLA

Purpose

To establish communication with the instrument. This function searches the VXI system for a module supported by the driver. It establishes communication with the module only if it is found at the specified Logical Address.

OBSERVATION B.3

Logical addresses are not guaranteed to be unique in a multi-mainframe system where slot numbers are not unique.

Parameters

Inputs	Description	Base Type
`logAdr`	VXI Logical Address	`ViInt16`

Outputs	Description	Base Type
`vi`	Instrument handle	`ViSession`

Return Values

Type `ViStatus`	Operational return status. Contains either a completion code or an error code. Instrument driver specific codes that may be returned in addition to the VISA error codes defined in VPP-4.3 and vendor specified codes, are as follows.

Completion Codes	Description
`VI_SUCCESS`	Initialization successful
`VI_MORE_INST_PRESENT`	One or more valid sessions was returned. However, there are additional instruments in the system not accessed by this function.

RULE B.5

IF the autoConnectToLA function encounters an error, **THEN** the value of the `vi` output parameter **SHALL** be `VI_NULL` and any valid sessions obtained from `viOpen` **SHALL** be closed.

RECOMMENDATION B.5

Any valid sessions obtained from viOpen that are subsequently closed should be equal to zero.

B.2.4 autoConnectToALL

Purpose

To establish communication with the instrument. autoConnectToAll attempts to find module(s) supported by the driver in the system. It will connect to all instances of the instrument found. If no instrument of this type is found autoConnectToAll will fail.

Parameters

Inputs	Description	Base Type
`arrayLength`	Size of instrument handle array	`ViInt16`

Outputs	Description	Base Type
`instrArray`	Array of session handles	`ViSession []`
`numConnected`	The number of instrument handles in the array	`ViInt16`

Return Values

Type `ViStatus`	Operational return status. Contains either a completion code or an error code. Instrument driver specific codes that may be returned in addition to the VISA error codes defined in VPP-4.3 and vendor specified codes, are as follows.

Completion Codes	Description
`VI_SUCCESS`	Initialization successful
`VI_MORE_INST_PRESENT`	One or more valid sessions was returned. However, there are additional instruments in the system not accessed by this function.

RULE B.6

IF the autoConnectToALL function encounters an error, **THEN** the value of the `numConnected` output parameter **SHALL** be zero and any valid sessions obtained from `viOpen` **SHALL** be closed.

RECOMMENDATION B.6

Any valid sessions obtained from viOpen that are subsequently closed should be equal to zero.

B.3 Bindings for C Language Optional Instrument Driver Functions

This section defines the function prototypes for the ANSI C bindings for the standard instrument driver functions from Section B.2.

```
ViStatus _VI_FUNC PREFIX_autoConnectToFirst(ViPSession vi);

ViStatus _VI_FUNC PREFIX_autoConnectToSlot(ViSession instrArray[],
                            ViInt16 arrayLength,
                            ViPInt16 numConnected, ViInt16 slot);
```

```
ViStatus _VI_FUNC PREFIX_autoConnectToLA(ViPSession vi, ViInt16 logAdr);

ViStatus _VI_FUNC PREFIX_autoConnectToAll(ViSession instrArray[],
                                ViInt16 arrayLength,
                                ViPInt16 numConnected);
```

RULE B.7

If VXI*plug&play* C language instrument driver implements any of the optional instrument driver functions, **THEN** the driver **SHALL** implement the functions with the ANSI C bindings defined in Section B.3.

PERMISSION B.1

A VXI*plug&play* C language instrument driver may use other names for parameters than those specified in Sections B.3.

RULE B.8

If VXI*plug&play* C language instrument driver implements any of the optional instrument driver functions, **THEN** the instrument driver header file (`PREFIX.h`) **SHALL** contain function prototypes for the functions as specified in Section B.3.