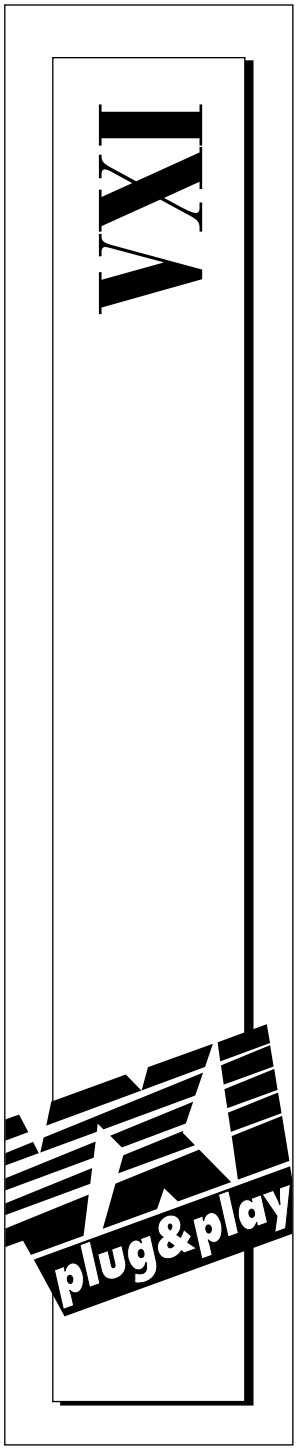

NI-VXI™ Programmer Reference Manual

July 1996 Edition
Part Number 321272A-01

Internet Support

GPIB: gpib.support@natinst.com
DAQ: daq.support@natinst.com
VXI: vxi.support@natinst.com
LabVIEW: lv.support@natinst.com
LabWindows: lw.support@natinst.com
HiQ: hiq.support@natinst.com
VISA: visa.support@natinst.com
Lookout: lookout.support@natinst.com

E-mail: info@natinst.com
FTP Site: ftp.natinst.com
Web Address: http://www.natinst.com

Bulletin Board Support

BBS United States: (512) 794-5422 or (800) 327-3077
BBS United Kingdom: 01635 551422
BBS France: 1 48 65 15 59

FaxBack Support

(512) 418-1111

Telephone Support (U.S.)

Tel: (512) 795-8248
Fax: (512) 794-5678

International Offices

Australia 03 9 879 9422, Austria 0662 45 79 90 0, Belgium 02 757 00 20,
Canada (Ontario) 519 622 9310, Canada (Québec) 514 694 8521, Denmark 45 76 26 00,
Finland 90 527 2321, France 1 48 14 24 24, Germany 089 741 31 30,
Hong Kong 2645 3186, Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456,
Mexico 95 800 010 0793, Netherlands 0348 433466, Norway 32 84 84 00,
Singapore 2265886, Spain 91 640 0085, Sweden 08 730 49 70,
Switzerland 056 200 51 51, Taiwan 02 377 1200, U.K. 01635 523545

National Instruments Corporate Headquarters

6504 Bridge Point Parkway Austin, TX 78730-5039 Tel: (512) 794-0100

Important Information

Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this manual is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Trademarks

WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

National Instruments products are not designed with components and testing intended to ensure a level of reliability suitable for use in treatment and diagnosis of humans. Applications of National Instruments products involving medical or clinical treatment can create a potential for accidental injury caused by product failure, or by errors on the part of the user or application designer. Any use or application of National Instruments products for or involving medical or clinical treatment must be performed by properly trained and qualified medical personnel, and all traditional medical safeguards, equipment, and procedures that are appropriate in the particular situation to prevent serious injury or death should always continue to be used when National Instruments products are being used. National Instruments products are NOT intended to be a substitute for any form of established process, procedure, or equipment used to monitor or safeguard human health and safety in medical or clinical treatment.

Table of Contents

About This Manual

Chapter 1
NI-VXI Group Overview

Chapter 2
Function Reference

About This Manual

This manual provides detailed features of the NI-VXI software and the VXI/VME function calls in the C/C++ and BASIC language. This manual is meant to be used with the *NI-VXI User Manual*.

Organization of This Manual

The *NI-VXI Programmer Reference Manual* is organized as follows:

- Chapter 1, *NI-VXI Group Overview*, summarizes the groups of NI-VXI functions. Most of the functional groups apply to VME as well as VXI, except as noted. General descriptions about each group may help you apply a function to your application programming environment.
- Chapter 2, *Function Reference*, contains a detailed description of each NI-VXI function. Functions are listed alphabetically and are explained at the C/C++ and BASIC syntax level. Checkboxes under a function name indicate its relationship to a VXI/VME device and a C/C++ and BASIC programming language.
- Appendix A, *Function Classification Reference*, contains two tables you can use as a quick reference. Table A-1, *Function Listing by Group*, lists the NI-VXI functions by their group association. This arrangement can help you determine easily which functions are available within each group. Table A-2, *Function Listing by Name*, lists each function alphabetically. You can refer to this table if you don't remember the group association of a particular function. Both tables use checkmarks to represent whether a VXI function also applies to VME and also whether it is associated with C/C++ and/or BASIC.
- Appendix B, *Customer Communication*,contains forms you can use to request help from National Instruments or to comment on our products and manuals.

- The *Glossary* contains an alphabetical list and description of terms used in this manual, including abbreviations, acronyms, and metric prefixes.

Conventions Used in This Manual

The following conventions are used in this manual:

bold — Bold text denotes parameters, menus, menu items, dialog box buttons or options, or error messages.

bold italic — Bold italic text denotes a note, caution, or warning.

`bold monospace` — Bold text in this font denotes the messages and responses that the computer automatically prints to the screen. This font also emphasizes lines of example code that are different from the other examples.

italic — Italic text denotes emphasis, a cross reference, or an introduction to a key concept.

`monospace` — Text in this font denotes the names of all VXI function calls, source code, sections of code, function syntax, parameter names, console responses, variable names and syntax examples.

In this manual, numbers are decimal unless noted as follows:

- Binary numbers are indicated by a -b suffix (for example, 11010101b).
- Octal numbers are indicated by an -o suffix (for example, 325o).
- Hexadecimal numbers are indicated by an -h suffix (for example, D5h).
- ASCII character and string values are indicated by double quotation marks (for example, "This is a string").
- Long values are indicated by an -L suffix (for example, 0x1111L).

Abbreviations, acronyms, metric prefixes, mnemonics, symbols, and terms are listed in the *Glossary*.

Related Documentation

The following documents contain information that you may find helpful as you read this manual:

- *IEEE Standard for a Versatile Backplane Bus: VMEbus*, ANSI/IEEE Standard 1014-1987
- *Multisystem Extension Interface Bus Specification*, Version 2.0
- VXI-1, *VXIbus System Specification*, Revision 1.4, VXIbus Consortium
- VXI-6, *VXIbus Mainframe Extender Specification*, Revision 1.0, VXIbus Consortium

Customer Communication

National Instruments wants to receive your comments on our products and manuals. We are interested in the applications you develop with our products, and we want to help if you have problems with them. To make it easy for you to contact us, this manual provides technical assistance numbers you may use to reach us to help you solve technical problems.

Additionally, a form is provided for you to comment on the product documentation. You may also refer to your getting started manual for forms to help you gather information necessary to help us solve technical problems you might have. Completing the forms before contacting National Instruments helps us help you better and faster. This information is in Appendix B, *Customer Communication*, at the end of this manual.

Chapter 1

NI-VXI Group Overview

This chapter summarizes the groups of NI-VXI functions. Most of the functional groups apply to VME as well as VXI, except as noted. General descriptions about each group may help you apply a function to your application programming environment.

Before using this manual, you might find it helpful to refer to the *NI-VXI User Manual.* For instance, Chapter 1, *Introduction to NI-VXI*, introduces you to the concepts of the NI-VXI software, VXI/VME, and MXI devices. Extensive discussion and descriptions are provided in the subsequent chapters of the *NI-VXI User Manual.*

Each function listed in Chapter 2 of this manual belongs to one of the following groups. To determine the group to which your selected function belongs, you can also refer to Appendix A, *Function Classification Reference*.

System Configuration Functions—These functions allow you to query and modify resource manager information. You can also search for devices using their names or other attributes.

Commander Word Serial Protocol Functions—Word serial communication is the minimal mode of communication between message-based devices within the Commander/Servant hierarchy. Commander Word Serial functions let the local CPU (the CPU on which the NI-VXI interface resides) perform message-based Commander Word Serial communication with its Servants. Word Serial communication does not apply to VME devices.

Servant Word Serial Protocol Functions—The local CPU (the CPU on which the NI-VXI interface resides) uses the Servant Word Serial functions to perform message-based Servant Word Serial communication with its Commander. Word Serial communication does not apply to VME devices.

High-Level VXI/VMEbus Access Functions—With the high-level access functions, you have direct access to the VXI/VMEbus address spaces. You can use these functions to read, write, and move blocks of data between any of the VXIbus address spaces. When execution speed is not a critical issue, these functions provide an easy-to-use interface.

Low-Level VXI/VMEbus Access Functions—Low-level access is the fastest access method for directly reading from or writing to any of the VXI/VMEbus address spaces.

Local Resource Access Functions—With these functions, you have access to miscellaneous local resources such as the local CPU VXI/VME register set, Slot 0 MODID operations, and the local CPU VXI/VME Shared RAM. These functions are useful for shared memory type communication, non-Resource Manager operation, and debugging purposes.

System Interrupt Handler Functions—With these functions, you can handle miscellaneous system conditions that can occur in the VXI/VME environment.

VXI/VME Interrupt Functions—VXI/VME interrupts are a basic form of asynchronous communication used by devices with interrupter support. These functions can specify the status/ID processing method, install interrupt service routines, and assert specified VXI/VME interrupt lines with a specified status/ID value.

VXI Signal Functions—With these functions, bus master devices can interrupt another device. The signal functions can specify the signal routing, manipulate the global signal queue, and wait for a particular signal value (or set of values) to be received.

VXI Trigger Functions—These functions provide a standard interface to source and accept any of the VXIbus TTL or ECL trigger lines. The trigger functions support all VXI-defined trigger protocols, with the actual capabilities dependent on the specific hardware platform. These functions do not apply to VME devices.

VXIbus Extender Functions—These functions can be used to dynamically reconfigure multi-mainframe transparent mapping of the VXI/VME interrupt and trigger lines and utility bus signals.

Chapter 2

Function Reference

This chapter contains a detailed description of each NI-VXI function. Functions are listed alphabetically and are explained at the C/C++ and BASIC syntax level. Checkboxes under a function name indicate its relationship to a VXI/VME device and a C/C++ and BASIC programming language. For a quick reference refer to Appendix A, *Function Classification Reference*.

AcknowledgeTrig

☑ VXI ☐ VME ☑ C/C++ ☑ BASIC

C/C++ Syntax

```
NIVXI_STATUS AcknowledgeTrig(INT16 controller, UINT16 line);
```

BASIC Syntax

```
FUNCTION AcknowledgeTrig% (BYVAL controller%, BYVAL line%)
```

Purpose

Acknowledges the specified TTL/ECL or external (GPIO) trigger on the specified controller.

Parameters

Name	Direction	Description
controller	Input	Controller on which to acknowledge trigger interrupt
line	Input	TTL, ECL, or external trigger line to acknowledge (See the *Description* section for value descriptions.)

Return Values

Return Values	Return Status Description
1	Successful, protocol has no need to acknowledge
0	Successful
-1	Unsupported function (no hardware support)
-2	Invalid **controller**
-3	Invalid **line**
-4	**line** not supported
-12	**line** not configured for sensing
-17	No trigger sensed
-18	**line** not configured for external SEMI-SYNC

Description

The TTL/ECL trigger interrupt handler is called after an TTL/ECL trigger is sensed. If the sensed protocol requires an acknowledge (ASYNC or SEMI-SYNC protocols), the application should call `AcknowledgeTrig` after performing any device-dependent operations. If you configured a trigger line using the `TrigAssertConfig` function to participate in external General Purpose Input/Output (GPIO) SEMI-SYNC acknowledging, you can use `AcknowledgeTrig` to manually acknowledge a pending external SEMI-SYNC trigger.

Note: ***If you are using*** `DefaultTrigHandler`***, the acknowledgment will be performed automatically.***

The following table lists possible values for the **line** parameter.

Value	Trigger Line
0 to 7	TTL trigger lines 0 to 7
8 to 13	ECL trigger lines 0 to 5
40 to 49	External source/destination (GPIOs 0 to 9)

C/C++ Example:

```
/* Acknowledge a trigger interrupt for TTL line 4 on the local CPU or
   the first remote controller. */

INT16          controller;
UINT16         line;
NIVXI_STATUS   ret;

controller = -1;
line = 4;
ret = AcknowledgeTrig (controller, line);
```

BASIC Example:

```
' Acknowledge the ECL trigger interrupt for line 2 on the local
' CPU or the first remote controller.

controller% = -1
line% = 10
ret% = AcknowledgeTrig% (controller%, line%)
```

AcknowledgeVXIint

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS AcknowledgeVXIint(INT16 controller, UINT16 level,
UINT32 *statusId);
```

BASIC Syntax

```
FUNCTION AcknowledgeVXIint% (BYVAL controller%, BYVAL level%,
statusId&)
```

Purpose

Performs an Interrupt Acknowledge (IACK) cycle on the specified controller for a particular VXI/VME interrupt level.

☞ **Note:** ***This function is only for debugging purposes.***

Parameters

Name	Direction	Description
controller	Input	Controller on which to perform an IACK cycle
level	Input	Interrupt level to acknowledge
statusId	Output	Status/ID obtained during IACK cycle

Return Values

Return Values	Return Status Description
0	IACK cycle completed successfully
-1	Unsupported function (no hardware support for IACK)
-2	Invalid **controller**
-3	Invalid **level**
-4	Bus error occurred during IACK cycle

Description

VXI/VME interrupts are automatically acknowledged when enabled by `EnableVXItoSignalInt` and `EnableVXIint`. Use this function to manually acknowledge VXI/VME interrupts that the local device is not enabled to receive.

C/C++ Example:

```
/* Acknowledge Interrupt 4 on the local CPU (or first extended
   controller). */

INT16          controller;
UINT16         level;
UINT32         statusId;
NIVXI_STATUS   ret;

controller = -1;
level = 4;
ret = AcknowledgeVXIint (controller, level, &statusId);
```

BASIC Example:

```
' Acknowledge Interrupt 4 on the local CPU (or first extended
' controller).
controller% = -1
level% = 4
ret% = AcknowledgeVXIint% (controller%, level%, statusId&)
```

AssertSysreset

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS AssertSysreset(INT16 controller, UINT16 resetmode);
```

BASIC Syntax

```
FUNCTION AssertSysreset% (BYVAL controller%, BYVAL resetmode%)
```

Purpose

Asserts the SYSRESET* signal on the specified controller.

Parameters

Name	Direction	Description
controller	Input	Controller on which to assert SYSRESET* -1 = Default controller -2 = All controllers
resetmode	Input	Mode of execution 0 = Use setting from VXIedit configuration 1 = Force link between SYSRESET* and local reset (SYSRESET* resets local CPU) 2 = Break link between SYSRESET* and local reset (SYSRESET* does *not* reset local CPU)

Return Values

Return Values	Return Status Description
0	SYSRESET* signal successfully asserted
-1	`AssertSysreset` not supported
-2	Invalid **controller**

Description

The controller will assert the SYSRESET* line on the backplane. In addition, this function can reset the local VXI/VME interface and/or the local CPU. Notice however, that not all platforms support resetting the local CPU. On platforms that do not support the resetting of the local CPU, the `resetmode` is ignored, but the `SYSRESET*` line is asserted.

C/C++ Example:

```
/* Assert SYSRESET* on the first remote controller (or local CPU)
   without changing the current configuration. */

INT16          controller;
UINT16         resetmode;
NIVXI_STATUS   ret;

controller = -1;
resetmode = 0;
ret = AssertSysreset (controller, resetmode);
```

BASIC Example:

```
' Assert SYSRESET* on the first remote controller (or local
' CPU) without changing the current configuration.

controller% = -1
resetmode% = 0
ret% = AssertSysreset% (controller%, resetmode%)
```

AssertVXIint

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS AssertVXIint(INT16 controller, UINT16 level, UINT32
statusId);
```

BASIC Syntax

```
FUNCTION AssertVXIint% (BYVAL controller%, BYVAL level%, BYVAL
statusId&)
```

Purpose

Asserts a VXI/VME interrupt line on the specified controller.

Parameters

Name	Direction	Description
controller	Input	Controller on which to assert interrupts
level	Input	Interrupt level to assert
statusId	Input	Status/ID to present during IACK cycle

Return Values

Return Values	Return Status Description
0	Interrupt line asserted successfully
-1	Unsupported function (no hardware support for VXI/VME interrupter)
-2	Invalid **controller**
-3	Invalid **level**
-5	VXI/VME interrupt still pending from previous `AssertVXIint`

Description

When the VXI/VME interrupt is acknowledged (a VXI/VME IACK cycle occurs), the specified status/ID is passed to the device that acknowledges the VXI/VME interrupt. The function does not wait for the IACK cycle.

C/C++ Example:

```
/*  Assert Interrupt 4 on the local CPU (or first extended
    controller) with status/ID of 0x1111. */

NIVXI_STATUS   ret;
INT16          controller;
UINT16         level;
UINT32         statusId;

controller = -1;
level = 4;
statusId = 0x1111L;
ret = AssertVXIint (controller, level, statusId);
```

BASIC Example:

```
' Assert Interrupt 4 on the local CPU (or first extended
' controller) with status/ID of &H1111&.

controller% = -1
level% = 4
statusId& = &H1111&
ret% = AssertVXIint% (controller%, level%, statusId&)
```

CloseVXIlibrary

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS CloseVXIlibrary(void);
```

BASIC Syntax

```
FUNCTION CloseVXIlibrary% ()
```

Purpose

Disables interrupts and frees dynamic memory allocated for the internal device information table.

Parameters

None

Return Values

Return Values	Return Status Description
0	NI-VXI library closed successfully
1	Successful; previous `InitVXIlibrary` calls still pending
-1	NI-VXI library was not open

Description

This function should be called before the application is closed.

C/C++ Example:

```
/* NI-VXI application shell program. */

main()
{
NIVXI_STATUS   ret;

ret = InitVXIlibrary();
if (ret < 0)
/* InitVXIlibrary failed. */;

/*
Application-specific program.
*/

ret = CloseVXIlibrary();
}
```

BASIC Example:

```
' NI-VXI application shell program.

ret% = InitVXIlibrary% ()
IF ret% < 0 THEN
' InitVXIlibrary failed.
END IF

' Application-specific program.

ret% = CloseVXIlibrary% ()
```

CreateDevInfo

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS CreateDevInfo (INT16 la);
```

BASIC Syntax

```
FUNCTION CreateDevInfo% (BYVAL la%)
```

Purpose

Allocates space in the device information table for a new entry with logical address **la**.

Parameters

Name	Direction	Description
la	Input	Logical address of device for which to create an entry in the NI-VXI Resource Manager table.

Return Values

Return Values	Return Status Description
0	Entry successfully created
-1	**la** already exists
-2	**la** out of range 0 to 511
-3	Dynamic memory allocation failure

Description

This function sets the fields in the device information table for the entry to default values (NULL or unasserted values).

C/C++ Example:

```
/* Create a new entry for pseudo logical address 298. */

NIVXI_STATUS   ret;
INT16          la;

la = 298;
ret = CreateDevInfo (la);
if (ret < 0)
         /* An error occurred creating new entry. */;
```

BASIC Example:

```
' Create a new entry for pseudo logical address 298.

la% = 298
ret% = CreateDevInfo% (la%)
IF ret% < 0 THEN
          ' Error creating new entry.
END IF
```

DeAssertVXIint

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS DeAssertVXIint(INT16 controller, UINT16 level);
```

BASIC Syntax

```
FUNCTION DeAssertVXIint% (BYVAL controller%, BYVAL level%)
```

Purpose

Asynchronously unasserts a VXI/VME interrupt line previously asserted by the function `AssertVXIint`.

☞ **Note:** ***This function is only for debugging purposes. Unasserting a VXI/VME interrupt can cause a violation of the VME and VXIbus specifications.***

Parameters

Name	Direction	Description
controller	Input	Controller on which to unassert interrupts
level	Input	Interrupt level to deassert

Return Values

Return Values	Return Status Description
0	Interrupt line deasserted successfully
-1	Unsupported function (no hardware support)
-2	Invalid **controller**
-3	Invalid **level**

Description

DeAssertVXIint causes the driver to stop driving the VXI/VME interrupt line without waiting for an IACK cycle.

C/C++ Example:

```
/* Unassert Interrupt 4 on the local CPU (or first remote controller). */

INT16          controller;
UINT16         level;
INT16          ret;

controller = -1;
level = 4;
ret = DeAssertVXIint (controller, level);
```

BASIC Example:

```
' Unassert Interrupt 4 on the local CPU (or first extended
' controller).

controller% = -1
level% = 4
ret% = DeAssertVXIint% (controller%, level%)
```

DefaultACfailHandler

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
void DefaultACfailHandler (INT16 controller);
```

BASIC Syntax

```
SUB DefaultACfailHandler (BYVAL controller%)
```

Purpose

This default handler simply increments the global variable `ACfailRecv`.

Parameters

Name	Direction	Description
controller	Input	Controller on which `ACfail` was asserted.

Return Values

None

DefaultBusErrorHandler

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
void DefaultBusErrorHandler (void);
```

BASIC Syntax

```
SUB DefaultBusErrorHandler ()
```

Purpose

This default handler simply increments the global variable `BusErrorRecv`.

Parameters

None

Return Values

None

DefaultSignalHandler

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
void DefaultSignalHandler (UINT16 sigval);
```

BASIC Syntax

```
SUB DefaultSignalHandler (BYVAL sigval%)
```

Purpose

Default handler that is called when a signal is received.

Parameters

Name	Direction	Description
sigval	Input	Actual 16-bit VXI signal

Return Values

None

Description

`DefaultSignalHandler` does nothing with the signals, with the exception of the VXIbus specification Revision 1.2 Event signal *Unrecognized Command*. It calls `WSabort` if the *Unrecognized Command* Event is received.

DefaultSoftResetHandler

☑ VXI ☐ VME ☑ C/C++ ☑ BASIC

C/C++ Syntax

```
void DefaultSoftResetHandler (void);
```

BASIC Syntax

```
SUB DefaultSoftResetHandler ()
```

Purpose

This default handler simply increments the global variable `SoftResetRecv`.

Parameters

None

Return Values

None

DefaultSysfailHandler

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
void DefaultSysfailHandler (INT16 controller);
```

BASIC Syntax

```
SUB DefaultSysfailHandler (BYVAL controller%)
```

Purpose

Handles the interrupt generated when the SYSFAIL* signal on the VXI/VME backplane is asserted.

Parameters

Name	Direction	Description
controller	Input	Controller on which `SYSFAIL` was asserted.

Return Values

None

Description

If a Servant is detected to have failed (as indicated when its PASS bit is cleared), the default Sysfail handler sets that Servant's Sysfail Inhibit bit and optionally sets its Reset bit. In addition, the global variable `SysfailRecv` is incremented.

DefaultSysresetHandler

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
void DefaultSysresetHandler (INT16 controller);
```

BASIC Syntax

```
SUB DefaultSysresetHandler (BYVAL controller%)
```

Purpose

Handles the interrupt generated when the SYSRESET* signal on the VXI/VME backplane is asserted (and the local CPU is not configured to be reset itself).

Parameters

Name	Direction	Description
controller	Input	Controller on which SYSRESET was asserted.

Return Values

None

Description

This default handler simply increments the global variable SysresetRecv.

DefaultTrigHandler

✔ VXI ☐ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
void DefaultTrigHandler (INT16 controller, UINT16 line, UINT16
type);
```

BASIC Syntax

```
SUB DefaultTrigHandler (BYVAL controller%, BYVAL line%, BYVAL
type%)
```

Purpose

Handles the VXI triggers on specified trigger lines.

Parameters

Name	Direction	Description
controller	Input	Controller on which to handle triggers
line	Input	Trigger line on which the interrupt is received (See the *Description* section for value descriptions.)
type	Input	Conditioning effect (See the *Description* section for possible bit values.)

Return Values

None

Description

Calls the `AcknowledgeTrig` function to acknowledge the trigger interrupt if the **type** parameter specifies trigger sensed. Otherwise, the interrupt is ignored.

The following table lists possible values for the **line** parameter.

Value	Trigger Line
0 to 7	TTL trigger lines 0 to 7
8 to 13	ECL trigger lines 0 to 5
50	TIC counter
60	TIC TICK1 tick timer

The following table describes the meaning of the bits for the **type** parameter.

Bit	Conditioning Effect
0	0 = Sourced trigger acknowledged 1 = Trigger sensed
2	1 = Assertion edge overrun occurred
3	1 = Unassertion edge overrun occurred
4	1 = Pulse stretch overrun occurred
15	1 = Error summary (2, 3, 4 = 1)

DefaultTrigHandler2

✔ VXI ☐ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
void DefaultTrigHandler2 (INT16 controller, UINT16 line, UINT16 type);
```

BASIC Syntax

```
SUB DefaultTrigHandler2 (BYVAL controller%, BYVAL line%, BYVAL
type%)
```

Purpose

Handles the VXI triggers on specified trigger lines.

Parameters

Name	Direction	Description
controller	Input	Controller on which the handle triggers
line	Input	Trigger line on which the interrupt is received (See the *Description* section for value descriptions.)
type	Input	Conditioning effect (See the *Description* section for possible bit values.)

Return Values

None

Description

This trigger interrupt handler performs no operations. Any triggers that require acknowledgments must be acknowledged by the user application.

The following table lists possible values for the **line** parameter.

Value	Trigger Line
0 to 7	TTL trigger lines 0 to 7
8 to 13	ECL trigger lines 0 to 5
50	TIC counter
60	TIC TICK1 tick timer

The following table describes the meaning of the bits for the **type** parameter.

Bit	Conditioning Effect
0	0 = Sourced trigger acknowledged 1 = Trigger sensed
2	1 = Assertion edge overrun occurred
3	1 = Unassertion edge overrun occurred
4	1 = Pulse stretch overrun occurred
15	1 = Error summary (2, 3, 4 = 1)

DefaultVXIintHandler

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
void DefaultVXIintHandler (INT16 controller, UINT16 level, UINT32
statusId);
```

BASIC Syntax

```
SUB DefaultVXIintHandler (BYVAL controller%, BYVAL level%,
BYVAL statusId&)
```

Purpose

Handles the VXI/VME interrupts.

Parameters

Name	Direction	Description
controller	Input	Controller on which to handle interrupts
level	Input	The received VXI/VME interrupt level
statusId	Input	Status/ID obtained during IACK cycle

Return Values

None

Description

The global variable `VXIintController` is set to **controller**. `VXIintLevel` is set to **level**. `VXIintStatusId` is set to **statusId**.

DefaultWSScmdHandler

✔ VXI ☐ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
void DefaultWSScmdHandler (UINT16 cmd);
```

Purpose

Handles any Word Serial Protocol command or query received from a VXI message-based Commander.

Parameters

Name	Direction	Description
cmd	Input	16-bit Word Serial command received

Return Values

None

Description

Uses global variables to handle many of the Word Serial commands. Implements all commands required for Servant operation.

DefaultWSSEcmdHandler

✔ VXI ☐ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
void DefaultWSSEcmdHandler (UINT16 cmdExt, UINT32 cmd);
```

Purpose

Handles Extended Longword Serial Protocol commands or queries received from a VXI message-based Commander.

Parameters

Name	Direction	Description
cmdExt	Input	Upper 16 bits of 48-bit Extended Longword Serial command received
cmd	Input	Lower 32 bits of 48-bit Extended Longword Serial command received

Return Values

None

Description

Returns an Unsupported Command protocol error for all commands and queries; the VXI specification does not define any Extended Longword Serial commands.

DefaultWSSLcmdHandler

✔ VXI ☐ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
void DefaultWSSLcmdHandler (UINT32 cmd);
```

Purpose

Handles Longword Serial Protocol commands or queries received from a VXI message-based Commander.

Parameters

Name	Direction	Description
cmd	Input	32-bit Longword Serial command received

Return Values

None

Description

Returns an Unsupported Command protocol error for all commands and queries; the VXI specification does not define any Longword Serial commands.

DefaultWSSrdHandler

☑ VXI ☐ VME ☑ C/C++ ☐ BASIC

C/C++ Syntax

```
void DefaultWSSrdHandler (INT16 status, UINT32 count);
```

Purpose

Handles the termination of a Servant Word Serial read operation started with `WSSrd`.

Parameters

Name	Direction	Description
status	Input	Status bit vector (See the *Description* section for possible bit values.)
count	Input	Actual number of bytes received

Return Values

None

Description

Sets the global variable `WSSrdDone` to 1, the `WSSrdDoneStatus` variable to **status**, and the `WSSrdDoneCount` global variable to **count**. The following table gives the meaning of each bit that is set to one (1) in the return value.

The following table lists possible bit values for the **status** parameter.

Bit	Name	Description
Bit 15 = 1 (Error Conditions)		
14	WRviol	Write Ready protocol violation during transfer
13	RRviol	Read Ready protocol violation during transfer
12	DORviol	Data Out Ready protocol violation
11	DIRviol	Data In Ready protocol violation
4	ForcedAbort	`WSSabort` called to force abort

Bit	Name	Description
Bit 15 = 0 (Successful Transfer)		
2	TC	All bytes received
1	END	END received with last byte
0	IODONE	Transfer successfully completed

DefaultWSSwrtHandler

✔ VXI ☐ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
void DefaultWSSwrtHandler (INT16 status, UINT32 count);
```

Purpose

Handles the termination of a Servant Word Serial write operation started with `WSSwrt`.

Parameters

Name	Direction	Description
status	Input	Status bit vector (See the *Description* section for possible bit values.)
count	Input	Actual number of bytes sent

Return Values

None

Description

Sets the global variable `WSSwrtDone` to 1, the `WSSwrtDoneStatus` variable to **status**, and the `WSSwrtDoneCount` variable to **count**. The following table gives the meaning of each bit that is set to one (1).

The following table lists possible bit values for the **status** parameter.

Bit	Name	Description
Bit 15 = 1 (Error Conditions)		
14	WRviol	Write Ready protocol violation during transfer
13	RRviol	Read Ready protocol violation during transfer
12	DORviol	Data Out Ready protocol violation
11	DIRviol	Data In Ready protocol violation
4	ForcedAbort	`WSSabort` called to force abort

Bit	Name	Description
Bit 15 = 0 (Successful Transfer)		
2	TC	All bytes sent
1	END	END sent with last byte
0	IODONE	Transfer successfully completed

DisableACfail

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS DisableACfail(INT16 controller);
```

BASIC Syntax

```
FUNCTION DisableACfail% (BYVAL controller%)
```

Purpose

Desensitizes the local CPU from interrupts generated from ACfail conditions on the specified controller.

Parameters

Name	Direction	Description
controller	Input	Controller on which to disable `ACfail` interrupt

Return Values

Return Values	Return Status Description
0	ACfail interrupt successfully disabled
-1	ACfail interrupts not supported
-2	Invalid **controller**

C/C++ Example:

```
/* Disable the ACfail interrupt on the first frame (or local CPU). */

INT16          controller;
NIVXI_STATUS   ret;

controller = -1;
ret = DisableACfail (controller);
```

BASIC Example:

```
' Disable the ACfail interrupt on the first frame (or local CPU).

controller% = -1
ret% = DisableACfail% (controller%)
```

DisableSignalInt

✔ VXI ☐ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS DisableSignalInt(void);
```

BASIC Syntax

```
FUNCTION DisableSignalInt% ()
```

Purpose

Desensitizes the local CPU to interrupts generated by writes to the local VXI Signal register.

Parameters

None

Return Values

Return Values	Return Status Description
0	Signal interrupts successfully disabled

Description

While signals are disabled, no VXI signals are processed. If the local VXI hardware Signal register is implemented as a FIFO, signals are held in the FIFO until the signal interrupt is enabled via the `EnableSignalInt` function. When the FIFO is full, the remote VXI device will get a bus error in response to a write to the Signal register.

C/C++ Example:

```
/* Disable the signal interrupt. */

NIVXI_STATUS   ret;

ret = DisableSignalInt ();
```

BASIC Example

```
' Disable the signal interrupt.

ret% = DisableSignalInt% ()
```

DisableSoftReset

✔ VXI ☐ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS DisableSoftReset(void);
```

BASIC Syntax

```
FUNCTION DisableSoftReset% ()
```

Purpose

Disables the local Soft Reset interrupt being generated from a write to the Reset bit of the local CPU Control register.

Parameters

None

Return Values

Return Values	Return Status Description
0	Soft Reset interrupt successfully disabled
-1	Soft Reset interrupts not supported

C/C++ Example:

```
/* Disable the Soft Reset interrupt. */
NIVXI_STATUS   ret;
ret = DisableSoftReset ();
```

BASIC Example:

```
' Disable the Soft Reset interrupt.
ret% = DisableSoftReset% ()
```

DisableSysfail

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS DisableSysfail(INT16 controller);
```

BASIC Syntax

```
FUNCTION DisableSysfail% (BYVAL controller%)
```

Purpose

Desensitizes the local CPU from interrupts generated from Sysfail conditions on the specified controller.

Parameters

Name	Direction	Description
controller	Input	Controller on which to disable `Sysfail` interrupt

Return Values

Return Values	Return Status Description
0	Sysfail interrupt successfully disabled
-1	Sysfail interrupts not supported
-2	Invalid **controller**

C/C++ Example:

```
/* Disable the Sysfail interrupt. */

INT16          controller;
NIVXI_STATUS   ret;
controller = -1;
ret = DisableSysfail();
```

BASIC Example:

```
' Disable the Sysfail interrupt.

controller% = -1
ret% = DisableSysfail% (controller%)
```

DisableSysreset

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS DisableSysreset(INT16 controller);
```

BASIC Syntax

```
FUNCTION DisableSysreset% (BYVAL controller%)
```

Purpose

Desensitizes the application from Sysreset interrupts on the specified controller.

Parameters

Name	Direction	Description
controller	Input	Controller on which to disable `Sysreset` interrupt

Return Values

Return Values	Return Status Description
0	Sysreset interrupt successfully disabled
-1	Sysreset interrupts not supported
-2	Invalid **controller**

C/C++ Example:

```
/* Disable the Sysreset interrupt. */

INT16          controller;
NIVXI_STATUS   ret;
controller = -1;
ret = DisableSysreset(controller);
```

BASIC Example:

```
' Disable the Sysreset interrupt.

controller% = -1
ret% = DisableSysreset% (controller%)
```

DisableTrigSense

✔ VXI ☐ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS DisableTrigSense(INT16 controller, UINT16 line);
```

BASIC Syntax

```
FUNCTION DisableTrigSense% (BYVAL controller%, BYVAL line%)
```

Purpose

Disables the sensing of the specified TTL/ECL trigger line, counter, or tick timer that was enabled by `EnableTrigSense`.

Parameters

Name	Direction	Description
controller	Input	Controller on which to disable sensing
line	Input	Trigger line to disable sensing (See the *Description* section for value descriptions.)

Return Values

Return Values	Return Status Description
0	Successful
-1	Unsupported function (no hardware support)
-2	Invalid **controller**
-3	Invalid **line**
-4	**line** not supported
-12	**line** not configured for sensing

Description

The following table lists possible values for the **line** parameter.

Value	Trigger Line
0 to 7	TTL trigger lines 0 to 7
8 to 13	ECL trigger lines 0 to 5
50	TIC counter
60	TIC tick timers

C/C++ Example:

```
/*  Disable sensing of TTL line 4 on the local CPU (or the first remote
    controller). */

NIVXI_STATUS   ret;
INT16          controller;
UINT16         line;

controller = -1;
line = 4;
ret = DisableTrigSense (controller, line);
```

BASIC Example:

```
' Disable sensing of ECL line 2 on the local CPU (or the first
' remote controller).

controller% = -1
line% = 10
ret% = DisableTrigSense% (controller%, line%)
```

DisableVXIint

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS DisableVXIint(INT16 controller, UINT16 levels);
```

BASIC Syntax

```
FUNCTION DisableVXIint% (BYVAL controller%, BYVAL levels%)
```

Purpose

Desensitizes the local CPU to specified VXI/VME interrupts generated in the specified controller, which the `RouteVXIint` function routed to be handled as VXI/VME interrupts (not as VXI signals).

Parameters

Name	Direction	Description
controller	Input	Controller on which to disable interrupts
levels	Input	Vector of VXI/VME interrupt levels to disable. Bits 6 to 0 correspond to the VXI/VME interrupt levels 7 to 1, respectively. 1 = Disable for appropriate level 0 = Leave at current setting

Return Values

Return Values	Return Status Description
0	VXI/VME interrupt disabled
-1	No hardware support
-2	Invalid **controller**

Description

The VXI Resource Manager assigns the interrupt levels automatically for programmable interrupter devices. `GetDevInfo` can be used to retrieve the assigned levels.

C/C++ Example:

```
/*  Disable VXI/VME Interrupt 4 on the local CPU (or first remote
    controller). */

INT16          controller;
UINT16         levels;
NIVXI_STATUS   ret;

controller = -1;   /** Local CPU or first frame. **/
levels = (UINT16)(1<<3);  /** Interrupt level 4. **/
ret = DisableVXIint (controller, levels);
```

BASIC Example:

```
' Disable VXI Interrupt 4 on the local CPU (or first extended
' controller).

controller% = -1   ' Local CPU or first frame.
levels% = &H0008   ' Interrupt level 4.
ret% = DisableVXIint% (controller%, levels%)
```

DisableVXItoSignalInt

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS DisableVXItoSignalInt(INT16 controller, UINT16
levels);
```

BASIC Syntax

```
FUNCTION DisableVXItoSignalInt% (BYVAL controller%, BYVAL levels%)
```

Purpose

Desensitizes the local CPU to specified VXI/VME interrupts generated in the specified controller, which the `RouteVXIint` function routed to be handled as VXI signals.

Parameters

Name	Direction	Description
controller	Input	Controller on which to disable interrupts
levels	Input	Vector of VXI/VME interrupt levels to disable. Bits 6 to 0 correspond to VXI/VME interrupt levels 7 to 1, respectively. 1 = Disable for appropriate level 0 = Leave at current setting

Return Values

Return Values	Return Status Description
0	VXI/VME interrupt disabled
-1	No hardware support
-2	Invalid **controller**

C/C++ Example:

```
/*  Disable VXI/VME Interrupt 6 on the local CPU (or first remote
    controller). */

INT16          controller;
UINT16         levels;
NIVXI_STATUS   ret;

controller = -1;   /** Local CPU or first frame. **/
levels = (UINT16)(1<<5);  /** Interrupt level 6. **/
ret = DisableVXItoSignalInt (controller, levels);
```

BASIC Example:

```
' Disable VXI Interrupt 6 on the local CPU (or first extended
' controller).

controller% = -1   ' Local CPU or first frame.
levels% = &H0020   ' Interrupt level 6.
ret% = DisableVXItoSignalInt% (controller%, levels%)
```

EnableACfail

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS EnableACfail(INT16 controller);
```

BASIC Syntax

```
FUNCTION EnableACfail% (BYVAL controller%)
```

Purpose

Sensitizes the local CPU to interrupts generated from ACfail conditions on the specified controller.

Parameters

Name	Direction	Description
controller	Input	Controller on which to enable `ACfail` interrupt

Return Values

Return Values	Return Status Description
0	ACfail interrupt successfully enabled
-1	ACfail interrupts not supported
-2	Invalid **controller**

C/C++ Example:

```
/* Enable the ACfail interrupt on the first frame (or local CPU). */
INT16          controller;
NIVXI_STATUS   ret;
controller = -1;
ret = EnableACfail (controller);
```

BASIC Example:

```
' Enable the ACfail interrupt on the first frame (or local CPU).

controller% = -1
ret% = EnableACfail% (controller%)
```

EnableSignalInt

✔ VXI ☐ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS EnableSignalInt(void);
```

BASIC Syntax

```
FUNCTION EnableSignalInt% ()
```

Purpose

Sensitizes the local CPU to interrupts generated by writes to the local VXI Signal register.

Parameters

None

Return Values

Return Values	Return Status Description
0	Signal interrupts successfully enabled
1	Signal queue full; will enable after dequeuing a signal

C/C++ Example:

```
/* Enable the signal interrupt. */

NIVXI_STATUS   ret;

ret = EnableSignalInt ();
```

BASIC Example

```
' Enable the signal interrupt.
ret% = EnableSignalInt% ()
```

EnableSoftReset

✔ VXI ☐ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS EnableSoftReset(void);
```

BASIC Syntax

```
FUNCTION EnableSoftReset% ()
```

Parameters

None

Purpose

Enables the local Soft Reset interrupt being generated from a write to the Reset bit of the local CPU Control register.

Return Values

Return Values	Return Status Description
0	Soft Reset interrupt successfully enabled
-1	Soft Reset interrupts not supported

C/C++ Example:

```
/* Enable the Soft Reset interrupt. */

NIVXI_STATUS   ret;

ret = EnableSoftReset ();
```

BASIC Example:

```
' Enable the Soft Reset interrupt.

ret% = EnableSoftReset% ()
```

EnableSysfail

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS EnableSysfail(INT16 controller);
```

BASIC Syntax

```
FUNCTION EnableSysfail% (BYVAL controller%)
```

Purpose

Sensitizes the local CPU to interrupts generated from Sysfail conditions on the specified controller.

Parameters

Name	Direction	Description
controller	Input	Controller on which to enable `Sysfail` interrupt

Return Values

Return Values	Return Status Description
0	Sysfail interrupt successfully enabled
-1	Sysfail interrupts not supported
-2	Invalid **controller**

C/C++ Example:

```
/* Enable the Sysfail interrupt in the local CPU (or first frame). */

INT16          controller;
NIVXI_STATUS   ret;
controller = -1;
ret = EnableSysfail (controller);
```

BASIC Example:

```
' Enable the Sysfail interrupt in the local CPU (or first frame).

controller% = -1
ret% = EnableSysfail% (controller%)
```

EnableSysreset

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS EnableSysreset(INT16 controller);
```

BASIC Syntax

```
FUNCTION EnableSysreset% (BYVAL controller%)
```

Purpose

Sensitizes the application to Sysreset interrupts from the specified controller.

Parameters

Name	Direction	Description
controller	Input	Controller on which to enable `Sysreset` interrupt

Return Values

Return Values	Return Status Description
0	Sysreset interrupt successfully enabled
-1	Sysreset interrupts not supported
-2	Invalid **controller**

C/C++ Example:

```
/* Enable the Sysreset interrupt in the local CPU (or first frame) */

INT16          controller;
NIVXI_STATUS   ret;
controller = -1;
ret = EnableSysreset (controller);
```

BASIC Example:

```
' Enable the Sysreset interrupt in the local CPU (or first frame).

controller% = -1
ret% = EnableSysreset% (controller%)
```

EnableTrigSense

☑ VXI ☐ VME ☑ C/C++ ☑ BASIC

C/C++ Syntax

```
NIVXI_STATUS EnableTrigSense(INT16 controller, UINT16 line, UINT16
prot);
```

BASIC Syntax

```
FUNCTION EnableTrigSense% (BYVAL controller%, BYVAL line%,
BYVAL prot%)
```

Purpose

Enables the sensing of the specified TTL/ECL trigger line or starts up the counter or tick timer for the specified protocol.

Parameters

Name	Direction	Description
controller	Input	Controller on which to enable sensing
line	Input	Trigger line to enable sensing (See the *Description* section for value descriptions.)
prot	Input	Protocol to use 0 = ON 1 = OFF 2 = START 3 = STOP 4 = SYNC 5 = SEMI-SYNC 6 = ASYNC

Return Values

Return Values	Return Status Description
0	Successful
-1	Unsupported function (no hardware support)
-2	Invalid **controller**
-3	Invalid **line** or **prot**
-4	**line** not supported
-5	**prot** not supported
-7	**line** already in use
-12	**line** not configured for use in sensing
-15	Counter/timer operation already in progress

Description

When the protocol is sensed, the corresponding trigger interrupt handler is invoked. Before starting up the counter or tick timers, you must first call either the `TrigCntrConfig` or the `TrigTickConfig` function, respectively.

The following table lists possible values for the **line** parameter.

Value	Trigger Line
0 to 7	TTL trigger lines 0 to 7
8 to 13	ECL trigger lines 0 to 5
50	TIC counter*
60	TIC tick timers*
* Only with controllers that have the TIC ASIC	

C/C++ Example:

```
/*  Enable sensing of TTL line 4 on the local CPU (or the first remote
    controller) for SEMI-SYNC protocol. */

NIVXI_STATUS   ret;
INT16          controller;
UINT16         line;
UINT16         prot;

controller = -1;
line = 4;
prot = 5;
ret = EnableTrigSense (controller, line, prot);
```

BASIC Example:

```
' Enable sensing of ECL line 2 on the local CPU (or the first
' remote controller) for SEMI-SYNC protocol.

controller% = -1
line% = 10
prot% = 5
ret% = EnableTrigSense% (controller%, line%, prot%)
```

EnableVXIint

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS EnableVXIint(INT16 controller, UINT16 levels);
```

BASIC Syntax

```
FUNCTION EnableVXIint% (BYVAL controller%, BYVAL levels%)
```

Purpose

Sensitizes the local CPU to specified VXI/VME interrupts detected at the controller, which the `RouteVXIint` function routed to be handled as VXI/VME interrupts (not as VXI signals).

Parameters

Name	Direction	Description
controller	Input	Controller on which to enable interrupts
levels	Input	Vector of VXI/VME interrupt levels to enable. Bits 6 to 0 correspond to VXI/VME interrupt levels 7 to 1, respectively. 1 = Enable for appropriate level 0 = Leave at current setting

Return Values

Return Values	Return Status Description
0	VXI/VME interrupt enabled
-1	No hardware support
-2	Invalid **controller**

Description

The Resource Manager assigns the interrupt levels automatically. Use the `GetDevInfo` functions to retrieve the assigned levels. Notice that each VXI/VME interrupt is physically enabled only if the `RouteVXIint` function has specified that the VXI/VME interrupt be routed to be handled as a VME/VXI interrupt (not as a VXI signal).

C/C++ Example:

```
/*  Enable VXI/VME Interrupt 4 on the local CPU (or first remote
    controller). */

INT16          controller;
UINT16         levels;
NIVXI_STATUS   ret;

controller = -1;   /** Local CPU or first frame. **/
levels = (UINT16)(1<<3);  /** Interrupt level 4. **/
ret = EnableVXIint (controller, levels);
```

BASIC Example:

```
' Enable VXI Interrupt 4 on the local CPU (or first remote controller).

controller% = -1   ' Local CPU or first frame.
levels% = &H0008   ' Interrupt level 4.
ret% = EnableVXIint% (controller%, levels%)
```

EnableVXItoSignalInt

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS EnableVXItoSignalInt(INT16 controller, UINT16
levels);
```

BASIC Syntax

```
FUNCTION EnableVXItoSignalInt% (BYVAL controller%, BYVAL levels%)
```

Purpose

Sensitizes the local CPU to specified VXI/VME interrupts detected at the specified controller, which the `RouteVXIint` function routed to be handled as VXI signals.

Parameters

Name	Direction	Description
controller	Input	Controller on which to enable interrupts
levels	Input	Vector of VXI/VME interrupt levels to enable. Bits 6 to 0 correspond to VXI/VME interrupt levels 7 to 1, respectively. 1 = Enable for appropriate level 0 = Leave at current setting

Return Values

Return Values	Return Status Description
1	Signal queue full, will enable after a `SignalDeq()`
0	VXI/VME interrupt enabled
-1	No hardware support
-2	Invalid **controller**

Description

The Resource Manager assigns the interrupt levels automatically. Use the `GetDevInfo` functions to retrieve the assigned levels. Notice that each VXI/VME interrupt is physically enabled only if the `RouteVXIint` function has specified that the VXI/VME interrupt be routed to be handled as a VXI signal.

C/C++ Example:

```
/*  Enable VXI Interrupt 6 on the local CPU (or first remote
    controller). */

INT16          controller;
UINT16         levels;
NIVXI_STATUS   ret;

controller = -1;   /** Local CPU or first frame. **/
levels = (UINT16)(1<<5);  /** Interrupt level 6. **/
ret = EnableVXItoSignalInt (controller, levels);
```

BASIC Example:

```
' Enable VXI Interrupt 6 on the local CPU (or first extended
' controller).

controller% = -1   ' Local CPU or first frame.
levels% = &H0020   ' Interrupt level 6.
ret% = EnableVXItoSignalInt% (controller%, levels%)
```

FindDevLA

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS FindDevLA (INT8 *namepat, INT16 manid,INT16
modelcode, INT16 devclass, INT16 slot, INT16 mainframe, INT16
cmdrla, INT16 *la);
```

Basic Syntax

```
FUNCTION FindDevLA% (BYVAL namepat$, BYVAL manid%, BYVAL
modelcode%, BYVAL devclass%, BYVAL slot%, BYVAL mainframe%, BYVAL
cmdrla%, la%)
```

Purpose

Finds a VXI device with the specified attributes in the device information table and returns its logical address.

Parameters

Name	Direction	Description
namepat	Input	Name Pattern. A partial name is acceptable. For example, when searching for a device named GPIB-VXI you can specify the letters GP.
manid	Input	VXI Manufacturer ID number
modelcode	Input	Manufacturer's 12-bit or 16-bit model number
devclass	Input	Device class of the device 0 = Memory Class device 1 = Extended Class device 2 = Message-Based device 3 = Register-Based device
slot	Input	Slot location of the device
mainframe	Input	Mainframe location of device (logical address of extender)
cmdrla	Input	Commander's logical address
la	Output	Logical address of the device found

Return Values

Return Values	Return Status Description
0	A device matching the specification was found
-1	No device matching the specification was found

Description

If the **namepat** parameter is "" or any other attribute is -1, that attribute is not used in the matching algorithm. For **namepat**, it accepts a partial name (for example, MIO-64 will match either a MIO-64XE-10 or a MIO-64E-1). If two or more devices match, the function returns the logical address of the first device found.

C/C++ Example:

```
/*  Find the logical address of a device with manid = 0xff6 (National
    Instruments) and modelcode = 0xf10 (MIO-64X3-10). */

NIVXI_STATUS   ret;
INT8           *namepat;
INT16          manid;
INT16          modelcode;
INT16          devclass;
INT16          slot;
INT16          mainframe;
INT16          cmdrla;
INT16          la;

namepat = "";
manid = 0xff6;
modelcode = 0xf10;
devclass = -1;
slot = -1;
mainframe = -1;
cmdrla = -1;
ret = FindDevLA (namepat, manid, modelcode, devclass, slot, mainframe,
cmdrla, &la);

if (ret < 0)
         /*  No device with manid = 0xff6 and modelcode = 0xf10 was
             found. */;

else
         /*  Device was found; logical address in la. */;
```

BASIC Example:

```
' Find the logical address of a device with manid = &HFF6
'(National Instruments) and modelcode = &HF10 (M10-64XE-10).

DIM namepat AS STRING * 13

namepat$ = ""
manid% = &HFF6
modelcode% = &HF10
devclass% = -1
mainframe% = -1
slot% = -1
cmdrla% = -1
ret% = FindDevLA% (namepat$, manid%, modelcode%, devclass%, mainframe%,
slot%, cmdrla%, la%)
IF ret% < 0 THEN
        ' No device with manid = &HFF6 and modelcode = &HF10 was found.
ELSE
        ' Device was found, logical address in la.
END IF
```

GenProtError

✔ VXI ☐ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_STATUS GenProtError(UINT16 proterr);
```

Purpose

Generates a Word Serial protocol error if one is not already pending.

Parameters

Name	Direction	Description
proterr	Input	Protocol error to generate

Return Values

Return Values	Return Status Description
0	Successful
1	Successful, but error will be ignored because a previous error is pending
-1	Servant Word Serial functions not supported

Description

`GenProtError` asserts the Response register bit ERR* if the value of the protocol error, **proterr**, is not -1. If **proterr** is -1, it deasserts the ERR* bit. If no previous error existed, it saves the **proterr** value for response to a future *Read Protocol Error* query via the function `RespProtError`. If a previous error does exist, the ERR* bit remains asserted but the protocol error specified by **proterr** is ignored.

Value	Protocol Error Description
ffffh	Clear any protocol error condition
fffdh	Multiple Query Error (MQE)
fffch	Unsupported Command (UnSupCom)
fffbh	Data In Ready violation (DIRviol)
fffah	Data Out Ready violation (DORviol)
fff9h	Read Ready violation (RRviol)
fff8h	Write Ready violation (WRviol)
others	Reserved

C/C++ Example:

```
/* Generate a protocol error of DORviol. */

NIVXI_STATUS    ret;
UINT16          proterr;

proterr = 0xfffa;
ret = GenProtError (proterr);
if (ret < 0)
          /* An error occurred in GenProtError. */;
```

GetACfailHandler

✔ VXI ✔ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_HACFAIL * GetACfailHandler(void);
```

Purpose

Returns the address of the current ACfail interrupt handler.

Parameters

None

Return Values

Return Values	Return Status Description
func	Pointer to the current ACfail handler (NULL = no support for ACfail interrupt handling).

C/C++ Example:

```
/* Get the address of the ACfail handler. */

NIVXI_HACFAIL      *func;

func = GetACfailHandler();
```

GetBusErrorHandler

✔ VXI ✔ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_HBUSERROR  *  GetBusErrorHandler(void);
```

Purpose

Returns the address of the current user bus error interrupt handler.

Parameters

None

Return Values

Return Values	Return Status Description
func	Pointer to the current bus error handler (`NULL` = no support for bus error interrupt handling).

C/C++ Example:

```
/* Get the address of the bus error handler. */

NIVXI_HBUSERROR    *func;

func = GetBusErrorHandler();
```

GetByteOrder

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS GetByteOrder(UINT32 windownum, UINT16 *ordermode);
```

BASIC Syntax

```
FUNCTION GetByteOrder% (BYVAL windownum&, ordermode%)
```

Purpose

Gets the byte/word order of data transferred into or out of the specified window.

Parameters

Name	Direction	Description
windownum	Input	Window number as returned from `MapVXIAddress`
ordermode	Output	Contains the byte/word ordering 0 = Motorola byte ordering 1 = Intel byte ordering

Return Values

Return Values	Return Status Description
0	Successful
1	Byte order returned successfully; same for all
-1	Invalid **windownum**

C/C++ Example:

```
/* Get the byte order for the specified window. */

NIVXI_STATUS   ret;
UINT32         windownum;
UINT16         ordermode;

/* Window value is set in MapVXIAddress. */

ret = GetByteOrder (windownum, &ordermode);
```

BASIC Example:

```
' Get the byte order for the specified window.

' Window value is set in MapVXIAddress.

ret% = GetByteOrder% (windownum&, ordermode%)
```

GetContext

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS GetContext(UINT32 windownum, UINT32 *context);
```

BASIC Syntax

```
FUNCTION GetContext% (BYVAL windownum&, context&)
```

Purpose

Gets the current hardware interface settings (context) for the specified window.

Parameters

Name	Direction	Description
windownum	Input	Window number as returned from `MapVXIAddress`
context	Output	Returned VXI/VME hardware access context

Return Values

Return Values	Return Status Description
0	Successful
-1	Invalid **windownum**

C/C++ Example:

```
/* Get or set the context for a windownum. */

NIVXI_STATUS   ret;
UINT32         windownum;
UINT32         context;

         /* Windownum ID set in MapVXIAddress call. */

ret = GetContext (windownum, &context);

         /* Change window settings as needed. */

ret = SetContext (windownum, context);
```

BASIC Example:

```
' Get or set the context for a window.

' Window ID set in MapVXIAddress call.

ret% = GetContext% (windownum&, context&)

' Change window settings as needed.
```

GetDevInfo

✔ VXI ✔ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_STATUS GetDevInfo(INT16 la, UINT16 field, void *fieldvalue);
```

Purpose

Gets device information about a specified device.

Parameters

Name	Direction	Description
la	Input	Logical address of device to get information about
field	Input	Field identification number (Refer to the table included in the *Description* section for more information.)
fieldvalue	Output	Information for that field (size dependent on field)

Return Values

Return Values	Return Status Description
0	The specified information was returned
-1	Device not found
-2	Invalid **field** specified

Description

The following table lists fields that can be read using this function.

Field	Description
0	Retrieve entire RM table entry for the specified device (structure of all of the following)
1	Device name
2	Commander's logical address
3	Mainframe
4	Slot
5	Manufacturer identification number
6	Manufacturer name
7	Model code
8	Model name
9	Device class
10	Extended subclass (if extended class device)
11	Address space used
12	Base of A24/A32 memory
13	Size of A24/A32 memory
14	Memory type and access time
15	Bit vector list of VXI/VME interrupter lines
16	Bit vector list of VXI/VME interrupt handler lines
17	Mainframe extender and controller information (See the following table for possible bit values and descriptions.)
18	Asynchronous mode control state
19	Response enable state
20	Protocols supported
21	Capability/status flags
22	Status state (Passed/Failed, Ready/Not Ready)

Bit	Description
15 and 14	Reserved
13	1 = Remote controller 0 = Not remote controller
12	1 = Child side extender 0 = Parent side extender
11	1 = Frame extender 0 = Not frame extender
10	1 = Extending controller
9	1 = Embedded controller
8	1 = External controller
7 to 0	Frame extender towards root frame

C/C++ Example:

```
/* Get the model code of a device at Logical Address 4. */

NIVXI_STATUS    ret;
INT16           la;
UINT16          field;
UINT16          fieldvalue;

la = 4;
field = 7;
ret = GetDevInfo (la, field, &fieldvalue);
if (ret < 0)
          /* Invalid logical address or field specified. */;
```

GetDevInfoLong

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS GetDevInfoLong (INT16 la, UINT16 field, UINT32
*longvalue);
```

BASIC Syntax

```
FUNCTION GetDevInfoLong% (BYVAL la%, BYVAL field%, longvalue&)
```

Purpose

Gets information about a specified device from the device information table.

Parameters

Name	Direction	Description
la	Input	Logical address of device to get information about
field	Input	Field identification number (See the *Description* section for possible field values.)
longvalue	Output	Information for that field

Return Values

Return Values	Return Status Description
0	The specified information was returned
-1	Device not found
-2	Invalid **field**

Description

This function is layered on top of `GetDevInfo` and returns only those fields that are 32-bit integers. The next table lists fields that can be read using this function.

Field	Description
12	Base of A24/A32 memory
13	Size of A24/A32 memory

C/C++ Example:

```
/* Get the A24 base of a device at Logical Address 4. */

NIVXI_STATUS    ret;
INT16           la;
UINT16          field;
UINT32          longvalue;

la = 4;
field = 12;
ret = GetDevInfoLong (la, field, &longvalue);
if (ret < 0)
          /* Invalid logical address or field specified. */;
```

BASIC Example:

```
' Get the A24 base of a device at Logical Address 4.

la% = 4
field% = 12
ret% = GetDevInfoLong% (la%, field%, longvalue&)
IF ret% < 0 THEN
          ' Invalid logical address or field specified.
END IF
```

GetDevInfoShort

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS GetDevInfoShort (INT16 la, UINT16 field, UINT16
*shortvalue);
```

BASIC Syntax

```
FUNCTION GetDevInfoShort% (BYVAL la%, BYVAL field%, shortvalue%)
```

Purpose

Gets information about a specified device from the device information table.

Parameters

Name	Direction	Description
la	Input	Logical address of device to get information about
field	Input	Field identification number (See the *Description* section for possible field values.)
shortvalue	Output	Information for that field

Return Values

Return Values	Return Status Description
0	The specified information was returned
-1	Device not found
-2	Invalid **field**

Description

This function is layered on top of `GetDevInfo` and returns only those fields that are 16-bit integers. The next table lists fields that can be read using this function.

Field	Description
2	Commander's logical address
3	Mainframe
4	Slot
5	Manufacturer identification number
7	Model code
9	Device class
10	Extended subclass (if extended class device)
11	Address space used
14	Memory type and access time
15	Bit vector list of VXI/VME interrupter lines
16	Bit vector list of VXI/VME interrupt handler lines
17	Mainframe extender and controller information (See the next table for possible bit values and descriptions.)
18	Asynchronous mode control state
19	Response enable state
20	Protocols supported
21	Capability/status flags
22	Status state (Passed/Failed, Ready/Not Ready)

Bit	Description
15 and 14	Reserved
13	1 = Remote controller 0 = Not remote controller
12	1 = Child side extender 0 = Parent side extender
11	1 = Frame extender 0 = Not frame extender
10	1 = Extending controller
9	1 = Embedded controller
8	1 = External controller
7 to 0	Frame extender towards root frame

C/C++ Example:

```
/* Get the model code of a device at Logical Address 4. */

NIVXI_STATUS    ret;
INT16           la;
UINT16          field;
UINT16          shortvalue;

la = 4;
field = 7;
ret = GetDevInfoShort (la, field, &shortvalue);
if (ret < 0)
          /* Invalid logical address or field specified. */;
```

BASIC Example:

```
' Get the model code of a device at Logical Address 4.

la% = 4
field% = 7
ret% = GetDevInfoShort% (la%, field%, shortvalue%)
IF ret% < 0 THEN
          ' Invalid logical address or field specified.
END IF
```

GetDevInfoStr

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS GetDevInfoStr (INT16 la, UINT16 field,
UINT8 *stringvalue);
```

BASIC Syntax

```
FUNCTION GetDevInfoStr% (BYVAL la%, BYVAL field%, BYVAL
stringvalue$)
```

Purpose

Gets information about a specified device from the device information table.

Parameters

Name	Direction	Description
la	Input	Logical address of device to get information about
field	Input	Field identification number (See the *Description* section for possible field values.)
stringvalue	Output	Buffer to receive information for that field

Return Values

Return Values	Return Status Description
0	The specified information was returned
-1	Device not found
-2	Invalid **field**

Description

This function is layered on top of `GetDevInfo` and returns only those fields that are character strings. The next table lists fields that can be retrieved using this function.

Field	Description
1	Device name
6	Manufacturer name
8	Model name

C/C++ Example:

```
/* Get the model name of a device at Logical Address 4. */

NIVXI_STATUS    ret;
INT16           la;
UINT16          field;
UINT8           stringvalue[14];

la = 4;
field = 8;
ret = GetDevInfoStr (la, field, stringvalue);
if (ret < 0)
/* Invalid logical address or field specified. */;
```

BASIC Example:

```
' Get the model name of a device at Logical Address 4.

DIM stringvalue as STRING*14
la% = 4
field% = 8
ret% = GetDevInfoStr% (la%, field%, stringvalue$)
IF ret% < 0 THEN
        ' Invalid logical address or field specified.
END IF
```

GetMyLA

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
INT16 GetMyLA (void);
```

BASIC Syntax

```
FUNCTION GetMyLA% ()
```

Purpose

Gets the logical address of the local VXI/VME device (the VXI/VME device on which this copy of the NI-VXI software is running).

Parameters

None

Return Values

Return Values	Return Status Description
la	Logical address of the local device

C/C++ Example:

```
/*   Get my logical address. */

INT16          la;

la = GetMyLA();
```

BASIC Example:

```
' Get my logical address.

la% = GetMyLA% ()
```

GetPrivilege

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS GetPrivilege (UINT32 windownum, UINT16 *priv);
```

BASIC Syntax

```
FUNCTION GetPrivilege% (BYVAL windownum&, priv%)
```

Purpose

Gets the current VXI/VME access privilege for the specified window.

Parameters

Name	Direction	Description
windownum	Input	Window number as returned from `MapVXIAddress`
priv	Output	Access Privilege 0 = Nonprivileged data access 1 = Supervisory data access 2 = Nonprivileged program access 3 = Supervisory program access 4 = Nonprivileged block access 5 = Supervisory block access

Return Values

Return Values	Return Status Description
0	Successful
-1	Invalid **windownum**

C/C++ Example:

```
/* Get the privilege for a window. */

NIVXI_STATUS   ret;
UINT32         windownum;
UINT16         priv;

          /* Window value is returned from MapVXIAddress. */

ret = GetPrivilege (windownum, &priv);
if (ret < 0)
          /* An error occurred in GetPrivilege. */;
```

BASIC Example:

```
' Get the privilege for a window.

' Windownum value is returned from MapVXIAddress.

ret% = GetPrivilege% (windownum&, priv%)
IF ret% < 0 THEN
          ' Error occurred in GetPrivilege.
END IF
```

GetSignalHandler

✔ VXI ✔ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_HSIGNAL * GetSignalHandler (INT16 la);
```

Purpose

Returns the address of the current signal handler for a specified logical address.

Parameters

Name	Direction	Description
la	Input	Logical address for which to find address of signal handler -2 = Unknown **la** (miscellaneous)

Return Values

Return Values	Return Status Description
func	Pointer to the current signal handler for the specified logical address (NULL = no support for signal handling).

C/C++ Example:

```
/*   Get the address of the signal handler for Logical Address 5. */

NIVXI_HSIGNAL   *func;
INT16           la;

la = 5;
func = GetSignalHandler (la);
```

GetSoftResetHandler

✔ VXI ☐ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_HSOFTRESET  *  GetSoftResetHandler(void);
```

Purpose

Returns the address of the current Soft Reset interrupt handler.

Parameters

None

Return Values

Return Values	Return Status Description
func	Pointer to the current Soft Reset handler (`NULL` = no support for Soft Reset interrupt handling).

C/C++ Example:

```
/* Get the address of the Soft Reset handler. */

NIVXI_HSOFTRESET   *func;

func = GetSoftResetHandler();
```

GetSysfailHandler

✔ VXI ✔ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_HSYSFAIL  *  GetSysfailHandler(void);
```

Purpose

Returns the address of the current Sysfail interrupt handler.

Parameters

None

Return Values

Return Values	Return Status Description
func	Pointer to the current Sysfail handler (`NULL` = no support for Sysfail interrupt handling).

C/C++ Example:

```
/* Get the address of the Sysfail handler. */

NIVXI_HSYSFAIL     *func;

func = GetSysfailHandler ();
```

GetSysresetHandler

✔ VXI ✔ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_HSYSRESET  *  GetSysresetHandler(void);
```

Purpose

Returns the address of the current SYSRESET* interrupt handler.

Parameters

None

Return Values

Return Values	Return Status Description
func	Pointer to the current Sysreset handler (`NULL` = no support for Sysreset interrupt handling).

C/C++ Example:

```
/* Get the address of the SYSRESET* handler. */

NIVXI_HSYSRESET    *func;

func = GetSysresetHandler();
```

GetTrigHandler

✔ VXI ☐ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_HTRIG * GetTrigHandler(UINT16 line);
```

Purpose

Returns the address of the current TTL/ECL trigger, counter, or tick timer interrupt handler for a specified trigger source.

Parameters

Name	Direction	Description
line	Input	TTL, ECL trigger line or counter/tick (See the *Description* section for value descriptions.)

Return Values

Return Values	Return Status Description
func	Pointer to the current trigger handler for the specified trigger line or counter/tick (`NULL` = no support for trigger interrupt handling).

Description

The following table lists possible values for the **line** parameter.

Value	Trigger Line
0 to 7	TTL trigger lines 0 to 7
8 to 13	ECL trigger lines 0 to 5
50	TIC counter
60	TIC tick timers

C/C++ Example:

```
/*  Get the address of the trigger interrupt handler for TTL trigger
    line 4. */

NIVXI_HTRIG     *func;
UINT16          line;

line = 4;
func = GetTrigHandler (line);
```

GetVXIbusStatus

✔ VXI ✔ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_STATUS GetVXIbusStatus(INT16 controller, BusStatus *status);
```

Purpose

Gets information about the state of the VXI/VMEbus in a specified controller.

Parameters

Name	Direction	Description
controller	Input	Controller on which to get information
status	Output	Structure containing VXI/VMEbus status

Structure Description

```
typedef struct BusStat {
INT16  BusError; /* 1 = Last access BERRed          */
INT16  Sysfail;  /* 1 = SYSFAIL* asserted           */
INT16  Acfail;   /* 1 = ACFAIL* asserted            */
INT16  SignalIn; /* Number of signals queued (not used in VME) */
INT16  VXIints;  /* Bit vector 1 = interrupt asserted       */
INT16  ECLtrigs; /* Bit vector 1 = trigger asserted (not used
                    in VME) */
INT16  TTLtrigs; /* Bit vector 1 = trigger asserted (not used
                    in VME) */
} BusStat;
```

A value of -1 returned in any of the fields signifies that there is no hardware support to retrieve information for that particular VXI/VMEbus state.

Return Values

Return Values	Return Status Description
0	Status information received successfully
-1	Unsupported function (no hardware support)
-2	Invalid **controller**

C/C++ Example:

```
/* Get the VXI/VMEbus status from local (or first) controller. */

NIVXI_STATUS   ret;
INT16          controller;
BusStatus      status;

controller = -1;
ret = GetVXIbusStatus (controller, &status);
if (ret < 0)
         /* An error occurred in GetVXIbusStatus. */;
```

GetVXIbusStatusInd

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS GetVXIbusStatusInd(INT16 controller, UINT16 field,
INT16 *status);
```

BASIC Syntax

```
FUNCTION GetVXIbusStatusInd% (BYVAL controller%, BYVAL field%,
status%)
```

Purpose

Gets information about the state of the VXI/VMEbus for the specified field in a particular controller.

Parameters

Name	Direction	Description
controller	Input	Controller on which to get information (-2 = OR of all controllers)
field	Input	Number of field to return information on. (See the field value description table in the *Description* section for possible values.)
status	Output	VXI/VMEbus status A value of -1 in any of the fields means that there is no hardware support for that particular state.

Return Values

Return Values	Return Status Description
0	Status information received successfully
-1	Unsupported function (no hardware support)
-2	Invalid **controller**
-3	Invalid **field**

Description

The following table defines the values of the **field** parameter.

Field Values	Description
1	**BusError;** /* 1 = Last access BERRed */
2	**Sysfail;** /* 1 = SYSFAIL* asserted */
3	**Acfail;** /* 1 = ACFAIL* asserted */
4	**SignalIn;** /* Number of signals queued */
5	**VXIints;** /* Bit vector 1 = interrupt asserted */
6	**ECLtrigs;** /* Bit vector 1 = trigger asserted */
7	**TTLtrigs;** /* Bit vector 1 = trigger asserted */

C/C++ Example:

```
/* Get the VXI/VMEbus status for Sysfail on local (or first) controller. */

NIVXI_STATUS    ret;
INT16           controller;
UINT16          field;
INT16           status;

controller = -1;
field = 2;
ret = GetVXIbusStatusInd (controller, field, &status);
if (ret < 0)
          /* An error occurred in GetVXIbusStatusInd. */;
```

BASIC Example:

```
' Get the VXI/VMEbus status for Sysfail on local (or first) controller.

cntroller% = -1
field% = 2
ret% = GetVXIbusStatusInd% (controller%, field%, status%)
IF ret% < 0 THEN
          ' Error in GetVXIbusStatusInd.
END IF
```

GetVXIintHandler

✔ VXI ✔ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_HVXIINT * GetVXIintHandler(UINT16 level);
```

Purpose

Returns the address of the current VXI/VME interrupt handler for a specified VXI/VMEbus interrupt level.

Parameters

Name	Direction	Description
level	Input	VXI/VME interrupt level associated with the handler

Return Values

Return Values	Return Status Description
func	Pointer to the current VXI/VME interrupt handler for the specified interrupt level (`NULL` = no support for VXI/VME interrupt handling).

C/C++ Example:

```
/* Get the address of the VXI/VME interrupt handler for VXI/VME
   interrupt level 4. */

NIVXI_HVXIINT   *func;
UINT16          level;

level = 4;
func = GetVXIintHandler (level);
```

GetWindowRange

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS GetWindowRange(UINT32 windownum, UINT32 *windowbase,
UINT32 *windowend);
```

BASIC Syntax

```
FUNCTION GetWindowRange% (BYVAL windownum&, windowbase&,
windowend&)
```

Purpose

Gets the range of addresses that a particular window, allocated with the `MapVXIAddress` function, can currently access within a particular VXI/VMEbus address space.

Parameters

Name	Direction	Description
windownum	Input	Window number obtained from `MapVXIAddress`
windowbase	Output	Base VXI/VME address
windowend	Output	End VXI/VME address

Return Values

Return Values	Return Status Description
0	Successful
-1	Invalid **windownum**

C/C++ Example:

```
/* Get the range for the window obtained from MapVXIAddress. */

UINT16          accessparms;
UINT32          address;
INT32           timo;
UINT32          windownum;
UINT32          windowbase;
UINT32          windowend;
NIVXI_STATUS    ret;
void            *addr;

accessparms = 1;
address = 0xc100L;
timo = 0L;
addr = MapVXIAddress (accessparms, address, timo, &windownum, &ret);
if (ret < 0)
{     /* Map failed; handle error. */;
}

ret = GetWindowRange (windownum, &windowbase, &windowend);
```

BASIC Example:

```
' Get the range for the window obtained from MapVXIAddress.

accessparms% = 1
address& = &HC100&
timo& = 0&
addr& = MapVXIAddress& (accessparms%, address&, timo&,
            windownum&, ret%)
IF ret% < 0 THEN
          ' Map failed; handle error.
END IF

ret% = GetWindowRange% (windownum&, windowbase&, windowend&)
```

GetWSScmdHandler

✔ VXI ☐ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_HWSSCMD * GetWSScmdHandler(void);
```

Purpose

Returns the address of the current WSScmd interrupt handler.

Parameters

None

Return Values

Return Values	Return Status Description
func	Pointer to the current WSScmd interrupt handler (NULL = no support for WSScmd interrupt handling).

C/C++ Example:

```
/* Get the address of the WSScmd handler. */

NIVXI_HWSSCMD  *func;

func = GetWSScmdHandler();
```

GetWSSEcmdHandler

✔ VXI ☐ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_HWSSECMD * GetWSSEcmdHandler(void);
```

Purpose

Returns the address of the current WSSEcmd interrupt handler.

Parameters

None

Return Values

Return Values	Return Status Description
func	Pointer to the current WSSEcmd interrupt handler (NULL = no support for WSSEcmd interrupt handling).

C/C++ Example:

```
/* Get the address of the WSSEcmd handler. */

NIVXI_HWSSECMD   *func;

func = GetWSSEcmdHandler();
```

GetWSSLcmdHandler

✔ VXI ☐ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_HWSSLCMD * GetWSSLcmdHandler(void);
```

Purpose

Returns the address of the current WSSLcmd interrupt handler.

Parameters

None

Return Values

Return Values	Return Status Description
func	Pointer to the current WSSLcmd interrupt handler (NULL = no support for WSSLcmd interrupt handling).

C/C++ Example:

```
/* Get the address of the WSSLcmd handler. */

NIVXI_HWSSLCMD   *func;

func = GetWSSLcmdHandler();
```

GetWSSrdHandler

✔ VXI ☐ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_HWSSRD * GetWSSrdHandler(void);
```

Purpose

Returns the address of the current WSSrd done notification interrupt handler.

Parameters

None

Return Values

Return Values	Return Status Description
func	Pointer to the current WSSrd done notification interrupt handler (NULL = no support for WSSrd done notification interrupt handling).

C/C++ Example:

```
/* Get the address of the WSSrd done notification handler. */

NIVXI_HWSSRD   *func;

func = GetWSSrdHandler();
```

GetWSSwrtHandler

✔ VXI ☐ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_HWSSWRT * GetWSSwrtHandler(void);
```

Purpose

Returns the address of the current `WSSwrt` done notification interrupt handler.

Parameters

None

Return Values

Return Values	Return Status Description
func	Pointer to the current `WSSwrt` done notification interrupt handler (`NULL` = no support for `WSSwrt` done notification interrupt handling).

C/C++ Example:

```
/* Get the address of the WSSwrt done notification handler. */

NIVXI_HWSSWRT  *func;

func = GetWSSwrtHandler();
```

InitVXIlibrary

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS InitVXIlibrary (void);
```

BASIC Syntax

```
FUNCTION InitVXIlibrary% ()
```

Purpose

Allocates and initializes the data structures required by the NI-VXI library functions.

Parameters

None

Return Values

Return Values	Return Status Description
0	NI-VXI library initialized
1	NI-VXI library already initialized (repeat call)
-1	NI-VXI library initialization failed

Description

This function reads the RM table file and copies all of the device information into data structures in local memory. It also performs other initialization operations, such as installing the default interrupt handlers and initializing their associated global variables.

C/C++ Example:

```
/* Initialize for using the library functions. */

main()
{
          NIVXI_STATUS    ret;

          ret = InitVXIlibrary();
          if (ret < 0)
          /* InitVXIlibrary failed. */;

          /*
          Application-specific program.
          */

          ret = CloseVXIlibrary();
}
```

BASIC Example:

```
' NI-VXI application shell program.

ret% = InitVXIlibrary% ()
IF ret% < 0 THEN
          ' InitVXIlibrary failed.
END IF

' Application-specific program.

ret% = CloseVXIlibrary% ()
```

MapECLtrig

✔ VXI ☐ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS MapECLtrig(INT16 extender, UINT16 lines, UINT16
directions);
```

BASIC Syntax

```
FUNCTION MapECLtrig% (BYVAL extender%, BYVAL lines%,
BYVAL directions%)
```

Purpose

Maps the specified ECL trigger lines for the specified mainframe in the specified direction (into or out of the mainframe).

Parameters

Name	Direction	Description
extender	Input	Mainframe extender for which to map ECL lines
lines	Input	Bit vector of ECL trigger lines. Bits 5 to 0 correspond to ECL lines 5 to 0, respectively. 1 = Enable for appropriate line 0 = Disable for appropriate line
directions	Input	Bit vector of directions for ECL lines. Bits 5 to 0 correspond to ECL lines 5 to 0, respectively. 1 = Into the mainframe 0 = Out of the mainframe

Return Values

Return Values	Return Status Description
0	Successful
-1	Unsupportable function (no hardware support)
-2	Invalid **extender**

C/C++ Example:

```
/* Map ECL lines 0 and 1 on the mainframe extender at Logical Address 5
   to go into the mainframe. */

INT16           extender;
UINT16          lines;
UINT16          directions;
NIVXI_STATUS    ret;

extender = 5;
lines = (UINT16)((1<<0) | (1<<1));  /** ECL lines 0 and 1. **/
directions = (UINT16)((1<<0) | (1<<1));
ret = MapECLtrig (extender, lines, directions);
```

BASIC Example:

```
' Map ECL lines 0 and 1 on the mainframe extender at Logical Address 5
' to go into the mainframe.

extender% = 5
lines% = &H003            ' ECL lines 0 and 1.
directions% = &H0003
ret% = MapECLtrig% (extender%, lines%, directions%)
```

MapTrigToTrig

✔ VXI ☐ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS MapTrigToTrig(INT16 controller, UINT16 srcTrig,
UINT16 destTrig, UINT16 mapmode);
```

BASIC Syntax

```
FUNCTION MapTrigToTrig% (BYVAL controller%, BYVAL srcTrig%,
BYVAL destTrig%, BYVAL mapmode%)
```

Purpose

Maps the specified TTL, ECL, Star X, Star Y, external connection (GPIO), or miscellaneous signal line to another.

Parameters

Name	Direction	Description
controller	Input	Controller on which to map signal lines
srcTrig	Input	Source line to map to destination (See the *Description* section for value descriptions.)
destTrig	Input	Destination line to map from source (See the *Description* section for value descriptions.)
mapmode	Input	Signal conditioning mode (0 = no conditioning) (See the *Description* section for possible bit values.)

Return Values

Return Values	Return Status Description
0	Successful
-1	Unsupported function, no mapping capability
-2	Invalid **controller**
-8	Unsupported **srcTrig**

Return Values	Return Status Description
-9	Unsupported **destTrig**
-10	Unsupported **mapmode**
-11	Already mapped, must use `UnMapTrigToTrig`

Description

The support actually present is completely hardware dependent and is reflected in the error status and in hardware-specific documentation.

The following table lists possible values for the **srcTrig** and **destTrig** parameters.

Value	Source or Destination Line
0 to 7	TTL trigger lines 0 to 7
8 to 13	ECL trigger lines 0 to 5
14 to 26	Star X lines 0 to 12 *
27 to 39	Star Y lines 0 to 12 *
40 to 49	External source/destination (GPIOs 0 to 9)
40	Front panel In (connector 1)
41	Front panel Out (connector 2)
42	ECL bypass from front panel
43	Connection to EXTCLK input pin
44 to 49	Hardware-dependent GPIOs 4 to 9
50	TIC counter pulse output (TCNTR)
51	TIC counter finished output (GCNTR)
60	TIC TICK1 tick timer output
61	TIC TICK2 tick timer output
* Star X and Star Y are not currently supported lines.	

The following table defines the meaning of the bits in the **mapmode** parameter.

Bit	Conditioning Effect
0	Synchronize with next CLK edge
1	Invert signal polarity
2	Pulse stretch to one CLK minimum
3	Use EXTCLK (not CLK10) for conditioning
All other values are reserved for future expansion.	

C/C++ Example:

```
/* Map TTL line 4 on the local CPU (or first extended controller) to go
   out of the front panel with no signal conditioning. */

INT16          controller;
UINT16         srcTrig;
UINT16         destTrig;
UINT16         mapmode;
NIVXI_STATUS   ret;

controller = -1;   /* Local CPU */
src = 4;           /* TTL line 4. **/
dest = 41;         /* Front panel out connector. **/
mapmode = 0;       /* No conditioning. */
ret = MapTrigToTrig (controller, srcTrig, destTrig, mapmode);
```

BASIC Example:

```
' Map TTL line 4 on the local CPU (or first extended controller)
' to go out of the front panel with no signal conditioning.

controller% = -1          ' Local CPU
srcTrig% = 4              ' TTL line 4.
destTrig% = 41            ' Front panel out connector.
mapmode% = 0              ' No conditioning.
ret% = MapTrigToTrig% (controller%, srcTrig%, destTrig%, mapmode%)
```

MapTTLtrig

✔ VXI ☐ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS MapTTLtrig(INT16 extender, UINT16 lines, UINT16
directions);
```

BASIC Syntax

```
FUNCTION MapTTLtrig% (BYVAL extender%, BYVAL lines%, BYVAL
directions%)
```

Purpose

Maps the specified TTL trigger lines for the specified mainframe in the specified direction (into or out of the mainframe).

Parameters

Name	Direction	Description
extender	Input	Mainframe extender for which to map TTL lines
lines	Input	Bit vector of TTL trigger lines. Bits 7 to 0 correspond to TTL lines 7 to 0, respectively. 1 = Enable for appropriate line 0 = Disable for appropriate line
directions	Input	Bit vector of directions for TTL lines. Bits 7 to 0 correspond to TTL lines 7 to 0, respectively. 1 = Into the mainframe 0 = Out of the mainframe

Return Values

Return Values	Return Status Description
0	Successful
-1	Unsupportable function (no hardware support)
-2	Invalid **extender**

C/C++ Example:

```
/*  Map TTL lines 4 and 5 on the mainframe extender at Logical Address 5
    to go out of the mainframe. */

INT16           extender;
UINT16          lines;
UINT16          directions;
NIVXI_STATUS    ret;

extender = 5;
lines = (UINT16)((1<<4) | (1<<5));  /** TTL lines 4, 5. **/
directions = (UINT16)0x0000;
ret = MapTTLtrig (extender, lines, directions);
```

BASIC Example:

```
'  Map TTL lines 4 and 5 on the mainframe extender at Logical Address 5
   to go out of the mainframe.

extender% = 5
lines% = &H0030           ' TTL lines 4, 5.
directions% = &H0
ret% = MapTTLtrig% (extender%, lines%, directions%)
```

MapUtilBus

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS MapUtilBus(INT16 extender, UINT16 modes);
```

BASIC Syntax

```
FUNCTION MapUtilBus% (BYVAL extender%, BYVAL modes%)
```

Purpose

Maps the specified VXI/VME utility bus signal for the specified mainframe into and/or out of the mainframe.

Parameters

Name	Direction	Description
extender	Input	Chassis extender for which to map utility bus signals
modes	Input	Bit vector of utility bus signals corresponding to the utility bus signals 1 = Enable for corresponding signal and direction 0 = Disable for corresponding signal and direction (See the bit description table in the *Description* section for possible values.)

Return Values

Return Values	Return Status Description
0	Successful
-1	Unsupportable function (no hardware support)
-2	Invalid **extender**

Description

The next table lists fields and bits that can be set using this function. The utility bus signals include ACfail, Sysfail, and Sysreset.

Bit	Utility Bus Signal and Direction
5	ACfail into the chassis
4	ACfail out of the chassis
3	Sysfail into the chassis
2	Sysfail out of the chassis
1	Sysreset into the chassis
0	Sysreset out of the chassis

C/C++ Example:

```
/* Map Sysfail into Mainframe 5.  Map Sysreset into and out of
   Mainframe 5.  Do not map ACfail at all. */

INT16          extender;
UINT16         modes;
NIVXI_STATUS   ret;

extender = 5;
modes = (UINT16)((1<<3) | (1<<1) | (1<<0));
ret = MapUtilBus (extender, modes);
```

BASIC Example:

```
'  Map Sysfail into Mainframe 5.  Map Sysreset into and out of
'  Mainframe 5. Do not map ACfail at all.

extender% = 5
modes% = &H000B
ret% = MapUtilBus% (extender%, modes%)
```

MapVXIAddress

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
void * MapVXIAddress(UINT16 accessparms, UINT32 address, INT32
timo, UINT32 *windownum, NIVXI_STATUS *ret);
```

BASIC Syntax

```
FUNCTION MapVXIAddress& (BYVAL accessparms%, BYVAL address&,
BYVAL timo&, windownum&, ret%)
```

Purpose

Sets up a window into one of the VXI/VME address spaces according to the access parameters specified, and returns a pointer to a local CPU address that accesses the specified VXI/VME address.

Parameters

Name	Direction	Description
accessparms	Input	See the *Description* section for possible bit values
address	Input	Address within A16, A24, or A32
timo	Input	Timeout (in milliseconds)
windownum	Output	Window number for use with other functions
ret	Output	Return Status

Return Values

☞ **Note:** ***To maintain compatibility and portability, the pointer obtained by calling this function should be used only with the functions*** `VXIpeek` ***and*** `VXIpoke`**.**

Return Values	Return Status Description
0	Map successful
-2	Invalid/unsupported **accessparms**
-3	Invalid **address**
-5	Byte order not supported
-6	Offset not accessible with this hardware
-7	Privilege not supported
-8	Window still in use; must use `UnMapVXIAddress`

Description

This function also returns the window ID associated with the window, which is used with all other low-level VXI/VMEbus access functions.

The following table lists possible bit values for the **accessparms** parameter.

Bit	Description
0 to 1	VXI/VME Address Space 1 = A16 2 = A24 3 = A32
2 to 4	Access Privilege 0 = Nonprivileged data access 1 = Supervisory data access 2 = Nonprivileged program access 3 = Supervisory program access 4 = Nonprivileged block access 5 = Supervisory block access

Bit	Description
5	0
6	Access Mode 0 = Access Only 1 = Owner Access
7	Byte Order 0 = Motorola 1 = Intel
8 to 15	0

C/C++ Example:

```
/* Get the local address pointer for address 0xc100 in the A16 space
   (base of Logical Address 4's VXI registers) with nonprivileged data
   and Motorola byte order.  Wait up to 5 seconds to get "Access Only"
   access to the window. */

UINT16          accessparms;
UINT32          address;
INT32           timo;
UINT32          windownum;
NIVXI_STATUS    ret;
void            *addr;

accessparms = 1;   /* A16, Motorola, nonprivileged data.          */
address = 0xc100L; /* Address = 0xC000 + 0x40 * Logical Address   */
timo = 5000L;      /* 5 seconds (5000 milliseconds)               */
addr = MapVXIAddress (accessparms, address, timo, &windownum, &ret);
if (ret < 0)
         /* Unable to get the pointer. */;
```

BASIC Example:

```
' Get the local address address for address &HC100& in the A16
' space (base of Logical Address 4's VXI registers) with
' nonprivileged data and Motorola byte order.  Wait up to 5
' seconds to get "Access Only" access to the window.

accessparms% = 1   ' A16, Motorola, nonprivileged data.
address& = &HC100& ' Address = &HC000 + &H40 * Logical Address
timo& = 5000&      ' 5 seconds (5000 milliseconds)
addr& = MapVXIAddress& (accessparms%, address&, timo&, windownum&,
   ret%)
IF ret% < 0 THEN
         ' Unable to get the address.
END IF
```

MapVXIAddressSize

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIXVI_STATUS MapVXIAddressSize(UINT32 size);
```

BASIC Syntax

```
FUNCTION MapVXIAddressSize% (BYVAL size&)
```

Purpose

Sets the default size for the window returned from `MapVXIAddress()`. This default size applies to all windows (A16, A24, and A32) and all processes calling `MapVXIAddress`. This function only provides the suggested window size to the NI-VXI driver. The size of the window mapped by `MapVXIAddress` may be different from the size provided by `MapVXIAddressSize`. You should still use `GetWindowRange` to find out the actual size of any window mapped.

Parameters

Name	Direction	Description
size	Input	Size in bytes of window to be mapped.

Return Values

Return Values	Return Status Description
0	Successful
-1	No hardware support for changing the mapped window size

C/C++ Example:

```
/* Set the size of future mapping to 1 MB */

NIVXI_STATUS ret;

ret = MapVXIAddressSize(0x100000);
```

BASIC Example:

```
ret% = MapVXIAddressSize(&H10000&)
```

MapVXIint

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS MapVXIint(INT16 extender, UINT16 levels,
UINT16 directions);
```

BASIC Syntax

```
FUNCTION MapVXIint% (BYVAL extender%, BYVAL levels%,
BYVAL directions%)
```

Purpose

Maps the specified VXI/VME interrupt levels for the specified mainframe in the specified direction (into or out of the mainframe).

Parameters

Name	Direction	Description
extender	Input	Mainframe extender for which to map VXI interrupt levels (C/C++, BASIC) Chassis extender for which to map VME interrupt levels (VME)
levels	Input	Bit vector of VXI/VME interrupt levels. Bits 6 to 0 correspond to VXI/VME interrupt levels 7 to 1 respectively. 1 = Enable for appropriate level 0 = Disable for appropriate level
directions	Input	Bit vector of directions for VXI/VME interrupt levels. Bits 6 to 0 correspond to VXI/VME interrupt levels 7 to 1, respectively. 1 = Into the mainframe 0 = Out of the mainframe

Return Values

Return Values	Return Status Description
0	Successful
-1	Unsupportable function (no hardware support)
-2	Invalid **extender**

C/C++ Example:

```
/*  Map VXI/VME interrupt levels 4 and 7 on the mainframe extender
    at Logical Address 5 to go out of the mainframe.  Map VXI/VME
    interrupt level 1 to go into the mainframe. */

INT16          extender;
UINT16         levels;
UINT16         directions;
NIVXI_STATUS   ret;

extender = 5;
levels = (UINT16)((1<<0) | (1<<3) | (1<<6)); /** Levels 1, 4, 7. **/
directions = (UINT16)(1<<0);          /* Level 1 only one in. */
ret = MapVXIint (extender, levels, directions);
```

BASIC Example:

```
' Map VXI/VME interrupt levels 4 and 7 on the mainframe
' extender at Logical Address 5 to go out of the
' mainframe.  Map VXI/VME interrupt level 1 to go into
' the mainframe.

extender% = 5
levels% = &H0049         ' Levels 1, 4, 7.
directions% = &H0001     ' Level 1 only one in.
ret% = MapVXIint% (extender%, levels%, directions%)
```

ReadMODID

✔ VXI ☐ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS ReadMODID(UINT16 *modid);
```

BASIC Syntax

```
FUNCTION ReadMODID% (modid%)
```

Purpose

Senses the MODID lines of the VXIbus backplane.

Parameters

Name	Direction	Description
modid	Output	Bit vector as described in the bit description table in the *Description* section.

Return Values

Return Values	Return Status Description
0	Successfully read MODID lines
-1	Not a Slot 0 device

Description

The next table lists bits that can be set using this function. This function applies only to the local device, which must be a Slot 0 device.

Bit	Description
12 to 0	MODID lines 12 to 0, respectively
13	MODID enable bit

C/C++ Example:

```
/* Read all the MODID lines 0 to 12. */

NIVXI_STATUS   ret;
UINT16         modid;

ret = ReadMODID (&modid);
if (ret < 0)
          /* An error occurred in ReadMODID. */;
```

BASIC Example:

```
' Read all the MODID lines 0 to 12.

ret% = ReadMODID% (modid%)
IF ret% < 0 THEN
          ' Error occurred in ReadMODID.
END IF
```

RespProtError

✔ VXI ☐ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_STATUS RespProtError(void);
```

Purpose

Responds to the Word Serial *Read Protocol Error* query with the last protocol error generated via the `GenProtError` function, and then unasserts the ERR* bit.

Parameters

None

Return Values

Return Values	Return Status Description
0	Successful
-1	Servant Word Serial functions not supported
-2	Response is still pending and a multiple query error is generated

C/C++ Example:

```
/* Respond to the Word Serial Read Protocol Error query. */

NIVXI_STATUS   ret;

ret = RespProtError ();
if (ret < 0)
         /* An error occurred in RespProtError. */;
```

RouteSignal

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS RouteSignal (INT16 la, UINT32 modemask);
```

BASIC Syntax

```
FUNCTION RouteSignal% (BYVAL la%, BYVAL modemask&)
```

Purpose

Specifies how each type of signal is to be processed for each logical address.

Parameters

Name	Direction	Description
la	Input	Logical address to set handler for (-1 = any known **la**)
modemask	Input	A bit vector that specifies whether each type of signal is enqueued or handled by the signal handler. A zero in any bit position causes signals of the associated type to be queued on the global signal queue. All other signals are handled by the signal handler. (See the *Description* section for more information.)

Return Values

Return Values	Return Status Description
0	Successful
-1	Invalid **la**

Description

A signal can be enqueued on a global signal queue (for later dequeuing via `SignalDeq`) or handled by an installed signal handler for the specified logical address.

The following tables lists possible bit values for the **la** parameter.

Bit	Event Signal
If **la** is a message-based device	
14	User-Defined events
13	VXI Reserved events
12	Shared Memory events
11	Unrecognized Command events
10	Request False (REQF) events
9	Request True (REQT) events
8	No Cause Given events

Bit	Response Signal
If **la** is a message-based device	
7	Unused
6	B14 (Reserved for future definition)
5	Data Out Ready (DOR)
4	Data In Ready (DIR)
3	Protocol Error (ERR)
2	Read Ready (RR)
1	Write Ready (WR)
0	Fast Handshake (FHS)

Bit	Type of Signal (Status/ID) Values
If **la** is *not* a message-based device	
15 to 8	Active high bit (if 1 in bits 15 to 8, respectively)
7 to 0	Active low bit (if 0 in bits 15 to 8, respectively)

C/C++ Example 1:

```
/*  Route signals for Logical Address 4 so that only REQT and REQF
    signals are enqueued on the signal queue, and the rest of the
    signals are handled by the signal handler. */

INT16          la;
UINT32         modemask;
NIVXI_STATUS   ret;

la = 4;
modemask = 0xf9ffL;
ret = RouteSignal (la, modemask);
```

C/C++ Example 2:

```
/*  Route register-based status/ID values for Logical Address 7 so that
    all status/IDs with a 0 in bits 15 to 12 are queued and all
    status/IDs with a 1 in bits 11 to 8 are handled by the signal
    handler. */

INT16          la;
UINT32         modemask;
NIVXI_STATUS   ret;

la = 7;
modemask = 0x0ff0L;
ret = RouteSignal (la, modemask);
```

BASIC Example 1:

```
' Route signals for Logical Address 4 so that only REQT and REQF
' signals are enqueued on the signal queue, and the rest of the
' signals are handled by the signal handler.

la% = 4
modemask& = &HF9FF&
ret% = RouteSignal% (la%, modemask&)
```

BASIC Example 2:

```
' Route register-based status/ID values for Logical Address 7 so
' that all status/IDs with a 0 in bits 15 to 12 are queued and all
' status/IDs with a 1 in bits 11 to 8 are handled by the signal
' handler.

la% = 7
modemask& = &H0FF0&
ret% = RouteSignal% (la%, modemask&)
```

RouteVXIint

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS RouteVXIint(INT16 controller, UINT16 Sroute);
```

BASIC Syntax

```
FUNCTION RouteVXIint% (BYVAL controller%, BYVAL Sroute%)
```

Purpose

Specifies whether to route the status/ID value retrieved from a VXI/VME interrupt acknowledge cycle to the VXI/VME interrupt handler or to the signal processing routine.

Parameters

Name	Direction	Description
controller	Input	Controller on which to specify route
Sroute	Input	A bit vector that specifies whether to handle a VXI/VME interrupt as a signal or route it to the VXI/VME interrupt handler routine (See the *Description* section for more information about bits.)

Return Values

Return Values	Return Status Description
0	Successful
-1	No hardware support
-2	Invalid **controller**

Description

RouteVXIint dynamically enables and disables the appropriate VXI/VME interrupts based on the current settings from calls to EnableVXIint and EnableVXItoSignalInt. For most VME systems, you will first call RouteVXIint to instruct your system to handle interrupts as VME interrupts.

The following table gives the meaning of the bits for the **Sroute** parameter.

Bit	Description
6 to 0	Bits correspond to VXI/VME interrupt levels 7 to 1, respectively. 1 = Handle VXI/VME interrupt for this level as a signal 0 = Handle VXI/VME interrupt as a VXI interrupt
14 to 8	Bits correspond to VXI interrupt levels 7 to 1, respectively. 1 = Route as 8-bit VME status/ID (ID will be set to 255) 0 = Route as 16-bit VXI status/ID

C/C++ Example:

```
/* Route VXI interrupts for level 4 (on the local controller) to the VXI
   interrupt handler and the rest of the levels to the signal processor. */

INT16          controller;
UINT16         Sroute;
NIVXI_STATUS   ret;

controller = -1;
Sroute = ~(1<<3);
ret = RouteVXIint (controller, Sroute);
```

BASIC Example:

```
' Route VXI interrupts for level 4 (on the local controller) to
' the VXI interrupt handler and the rest of the levels to the
' signal processor.

controller% = -1
Sroute% = &HFFF7
ret% = RouteVXIint% (controller%, Sroute%)
```

SetACfailHandler

✔ VXI ✔ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_STATUS SetACfailHandler(NIVXI_HACFAIL *func);
```

Purpose

Replaces the current ACfail callback handler with a specified handler.

Parameters

Name	Direction	Description
func	Input	Pointer to the new ACfail callback handler. Specifying `NULL` will restore the `DefaultACfailHandler`.

Return Values

Return Values	Return Status Description
0	Successful
-1	ACfail interrupt not supported

C/C++ Example:

```
/* Set the ACfail handler. */

NIVXI_HACFAIL   func;
NIVXI_STATUS    ret;

ret = SetACfailHandler (func);
if (ret < 0)
         /* An error occurred in SetACfailHandler. */;


/* Example handler */
NIVXI_HQUAL void NIVXI_HSPEC func (INT16 controller)
{
}
```

SetBusErrorHandler

✔ VXI ✔ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_STATUS SetBusErrorHandler(NIVXI_HBUSERROR *func);
```

Purpose

Replaces the current bus error handler with a specified handler.

Parameters

Name	Direction	Description
func	Input	Pointer to the new bus error callback handler. Specifying `NULL` will restore the `DefaultBusErrorHandler`.

Return Values

Return Values	Return Status Description
0	Successful

C/C++ Example:

```
/* Set the bus error handler. */


NIVXI_HBUSERROR  func;
NIVXI_STATUS     ret;

ret = SetBusErrorHandler(func);
if (ret < 0)
         /* An error occurred in SetBusErrorHandler. */;


/* Example handler */
NIVXI_HQUAL void NIVXI_HSPEC func (void)
{
}
```

SetByteOrder

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS SetByteOrder(UINT32 windownum, UINT16 ordermode);
```

BASIC Syntax

```
FUNCTION SetByteOrder% (BYVAL windownum&, BYVAL ordermode%)
```

Purpose

Sets the byte/word order of data transferred into or out of the specified window.

Parameters

Name	Direction	Description
windownum	Input	Window number as returned from `MapVXIAddress`
ordermode	Input	Sets the byte/word ordering 0 = Motorola byte ordering 1 = Intel byte ordering

Return Values

Return Values	Return Status Description
0	Successful; byte order set for specific window only
1	Successful; byte order set for all windows
-1	Invalid **windownum**
-2	Invalid **ordermode**
-5	**ordermode** not supported
-9	**windownum** is not Owner Access

C/C++ Example:

```
/* Set the byte order to Motorola for a window. */

NIVXI_STATUS   ret;
UINT32         windownum;
UINT16         ordermode;

/* Window set in call to MapVXIAddress(). */
ordermode = 0;
ret = SetByteOrder (windownum, ordermode);
if (ret <0)
    /* An error occurred in SetByteOrder. */;
```

BASIC Example:

```
' Set the byte order to Motorola for a window.

' Window set in call to MapVXIAddress().
ordermode% = 0
ret% = SetByteOrder% (windownum&, ordermode%)
IF ret% < 0 THEN
    ' Capability not present.
END IF
```

SetContext

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS SetContext(UINT32 windownum, UINT32 context);
```

BASIC Syntax

```
FUNCTION SetContext% (BYVAL windownum&, BYVAL context&)
```

Purpose

Sets the current hardware interface settings (context) for the specified window.

Parameters

Name	Direction	Description
windownum	Input	Window number as returned from `MapVXIAddress`
context	Input	VXI/VME hardware context to install (context returned from `GetContext`)

Return Values

Return Values	Return Status Description
0	Successful
-1	Invalid **windownum**
-2	Invalid/unsupported **context**
-9	**windownum** is not Owner Access
-10	Base address change is not supported

Description

The value for **context** should have been set previously by the function `GetContext`.

C/C++ Example:

```
/* Get or set the context for a window. */

NIVXI_STATUS   ret;
UINT32         windownum;
UINT32         context;

          /* Window ID set in MapVXIAddress call. */
ret = GetContext (windownum, &context);

          /* Change window settings as needed. */

ret = SetContext (windownum, context);
if (ret <0)
    /* An error occurred in SetContext. */;
```

BASIC Example:

```
' Get or set the context for a window.

' Window ID set in MapVXIAddress call.
ret% = GetContext% (windownum&, context&)

' Change window settings as needed.

ret% = SetContext% (windownum&, context&)
```

SetDevInfo

✔ VXI ✔ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_STATUS SetDevInfo (INT16 la, UINT16 field, void
*fieldvalue);
```

Purpose

Sets information about a specified device in the device information table.

Name	Direction	Description
la	Input	Logical address of device for which to set information
field	Input	Field identification number (See the *Description* section for possible field values.)
fieldvalue	Input	Information for that field (size dependent on field)

Return Values

Return Values	Return Status Description
0	The specified information was returned
-1	Device not found
-2	Invalid **field** specified

Description

The following table defines the value of the **field** parameter.

Field	Description
0	Modify entire RM table entry for the specified device (structure of all of the following)
1	Device name
2	Commander's logical address
3	Mainframe
4	Slot
5	Manufacturer identification number
6	Manufacturer name
7	Model code
8	Model name
9	Device class
10	Extended subclass (if extended class device)
11	Address space used
12	Base of A24/A32 memory
13	Size of A24/A32 memory
14	Memory type and access time
15	Bit vector list of VXI/VME interrupter lines
16	Bit vector list of VXI/VME interrupt handler lines
17	Mainframe extender and controller information (See the following table for possible bit values and descriptions.)
18	Asynchronous mode control state
19	Response enable state
20	Protocols supported
21	Capability/status flags
22	Status state (Passed/Failed, Ready/Not Ready)

Bit	Description
15 and 14	Reserved
13	1 = Remote controller 0 = Not remote controller
12	1 = Child side extender 0 = Parent side extender
11	1 = Frame extender 0 = Not frame extender
10	1 = Extending controller
9	1 = Embedded controller
8	1 = External controller
7 to 0	Frame extender towards root frame

C/C++ Example:

```
/* Set the model code of a device at Logical Address 4. */

NIVXI_STATUS   ret;
INT16          la;
UINT16         field;
UINT32         fieldvalue;

la = 4;
field = 7;
fieldvalue = 0xffffL;
ret = SetDevInfo (la, field, &fieldvalue);
if (ret < 0)
         /* Invalid logical address or field specified. */;
```

SetDevInfoLong

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS SetDevInfoLong (INT16 la, UINT16 field, UINT32
longvalue);
```

BASIC Syntax

```
FUNCTION SetDevInfoLong% (BYVAL la%, BYVAL field%,
BYVAL longvalue&)
```

Purpose

Sets information about a specified device in the device information table.

Parameters

Name	Direction	Description
la	Input	Logical address of device for which to set information
field	Input	Field identification number (See the *Description* section for possible field values.)
longvalue	Input	Information for that field

Return Values

Return Values	Return Status Description
0	The specified information was returned
-1	Device not found
-2	Invalid **field**

Description

This function is layered on top of `SetDevInfo` and changes only those fields that are 32-bit integers. The following table defines the value of the **field** parameter.

Field	Description
12	Base of A24/A32 memory
13	Size of A24/A32 memory

C/C++ Example:

```
/* Set the A24 base of a device at Logical Address 4. */

NIVXI_STATUS   ret;
INT16          la;
UINT16         field;
UINT32         longvalue;

la = 4;
field = 12;
longvalue = 0x200000L;
ret = SetDevInfoLong (la, field, longvalue);
if (ret < 0)
/* Invalid logical address or field specified. */;
```

BASIC Example:

```
' Set the A24 base of a device at Logical Address 4.

la% = 4
field% = 12
longvalue& = &H200000&
ret% = SetDevInfoLong% (la%, field%, longvalue&)
IF ret% < 0 THEN
        ' Invalid logical address or field specified.
END IF
```

SetDevInfoShort

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS SetDevInfoShort (INT16 la, UINT16 field, UINT16
shortvalue);
```

BASIC Syntax

```
FUNCTION SetDevInfoShort% (BYVAL la%, BYVAL field%,
BYVAL shortvalue%)
```

Purpose

Sets information about a specified device in the device information table.

Parameters

Name	Direction	Description
la	Input	Logical address of device for which to set information
field	Input	Field identification number (See *Description* section for all possible field values.)
shortvalue	Input	Information for that field

Return Values

Return Values	Return Status Description
0	The specified information was returned
-1	Device not found
-2	Invalid **field**

Description

This function is layered on top of `SetDevInfo` and changes only those fields that are 16-bit integers. The following table defines the value of the **field** parameter.

Field	Description
2	Commander's logical address
3	Mainframe
4	Slot
5	Manufacturer identification number
7	Model code
9	Device class
10	Extended subclass (if extended class device)
11	Address space used
14	Memory type and access time
15	Bit vector list of VXI/VME interrupter lines
16	Bit vector list of VXI/VME interrupt handler lines
17	Mainframe extender and controller information (See the following table for possible bit values and descriptions.)
18	Asynchronous mode control state
19	Response enable state
20	Protocols supported
21	Capability/status flags
22	Status state (Passed/Failed, Ready/Not Ready)

Bit	Description
15 and 14	Reserved
13	1 = Remote controller 0 = Not remote controller
12	1 = Child side extender 0 = Parent side extender
11	1 = Frame extender 0 = Not frame extender
10	1 = Extending controller
9	1 = Embedded controller
8	1 = External controller
7 to 0	Frame extender towards root frame

C/C++ Example:

```
/* Set the model code of a device at Logical Address 4. */

NIVXI_STATUS   ret;
INT16          la = 4;
UINT16         field = 7;
UINT16         shortvalue = 0xffff;
ret = SetDevInfoShort (la, field, shortvalue);
if (ret < 0)
         /* Invalid logical address or field specified. */;
```

BASIC Example:

```
' Set the model code of a device at Logical Address 4.
la% = 4
field% = 7
shortvalue% = &HFFFF
ret% = SetDevInfoShort% (la%, field%, shortvalue%)
IF ret% < 0 THEN
         ' Invalid logical address or field specified.
END IF
```

SetDevInfoStr

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax:

```
NIVXI_STATUS SetDevInfoStr (INT16 la, UINT16 field, UINT8
*stringvalue);
```

BASIC Syntax:

```
FUNCTION SetDevInfoStr% (BYVAL la%, BYVAL field%, BYVAL
stringvalue$);
```

Purpose

Sets information about a specified device in the device information table.

Parameters

Name	Direction	Description
la	Input	Logical address of device for which to set information
field	Input	Field identification number (See the *Description* section for possible field values.)
stringvalue	Input	Buffer that contains new information for that field

Return Values

Return Values	Return Status Description
0	The specified information was returned
-1	Device not found
-2	Invalid **field**

Description

This function is layered on top of `SetDevInfo` and changes only those fields that are character strings. The following table defines the value of the **field** parameter.

Field	Description
1	Device name
6	Manufacturer name
8	Model name

C/C++ Example:

```
/* Set the model name of a device at Logical Address 4. */

NIVXI_STATUS   ret;
INT16          la;
UINT16         field;
UINT8          stringvalue[14];

la = 4;
field = 8;
strcpy (stringvalue, "DMM0");
ret = SetDevInfoStr (la, field, stringvalue);
if (ret < 0)
         /* Invalid logical address or field specified. */;
```

BASIC Example:

```
' Set the model name of a device at Logical Address 4.

la% = 4
field% = 8
stringvalue$ = "DMM0"
ret% = SetDevInfoStr% (la%, field%, stringvalue$)
IF ret% <> 0 THEN
          ' Invalid logical address or field specified.
END IF
```

SetMODID

✔ VXI ☐ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS SetMODID(UINT16 enable, UINT16 modid);
```

BASIC Syntax

```
FUNCTION SetMODID% (BYVAL enable%, BYVAL modid%)
```

Purpose

Controls the assertion of the MODID lines of the VXIbus backplane.

Parameters

Name	Direction	Description
enable	Input	1 = Set MODID enable bit 0 = Clear MODID enable bit
modid	Input	Bit vector for Bits 0 to 12, corresponding to Slots 0 to 12

Return Values

Return Values	Return Status Description
0	Successfully set MODID lines
-1	Not a Slot 0 device

Description

This function only applies to the local device, which must be a Slot 0 device.

C/C++ Example:

```
/* Set all the MODID lines 0 to 12. */

NIVXI_STATUS   ret;
UINT16         enable;
UINT16         modid;

enable = 1;
modid = 0x1fff;    /* Bit vector (Bits 0 to 12). */

ret = SetMODID (enable, modid);
if (ret < 0)
         /* An error occurred in SetMODID. */;
```

BASIC Example:

```
' Set all the MODID lines 0 to 12.

enable% = 1
modid% = &H1FFF    ' Bit vector (Bits 0 to 12).

ret% = SetMODID% (enable%, modid%)
IF ret% < 0 THEN
         ' Error occurred in SetMODID.
END IF
```

SetPrivilege

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS SetPrivilege(UINT32 windownum, UINT16 priv);
```

BASIC Syntax

```
FUNCTION SetPrivilege% (BYVAL windownum&, BYVAL priv%)
```

Purpose

Sets the VXI/VME access privilege for the specified window to the specified privilege state.

Parameters

Name	Direction	Description
windownum	Input	Window number as returned from `MapVXIAddress`
priv	Input	Access Privilege 0 = Nonprivileged data access 1 = Supervisory data access 2 = Nonprivileged program access 3 = Supervisory program access 4 = Nonprivileged block access 5 = Supervisory block access

Return Values

Return Values	Return Status Description
0	Successful
-1	Invalid **windownum**
-2	Invalid **priv**
-7	**priv** not supported
-9	**windownum** is not Owner Access

C/C++ Example:

```
/* Set nonprivileged data access for a window. */

NIVXI_STATUS    ret;
UINT32          windownum;
UINT16          priv;

/* Window ID set in MapVXIAddress call. */
priv = 0;
ret = SetPrivilege (windownum, priv);
if (ret < 0)
          /* An error occurred in SetPrivilege. */;
```

BASIC Example:

```
' Set nonprivileged data access for a window.

' Window ID set in MapVXIAddress call.
priv% = 0
ret% = SetPrivilege% (windownum&, priv%)
IF ret% < 0 THEN
          ' Error occurred in SetPrivilege.
END IF
```

SetSignalHandler

✔ VXI ✔ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_STATUS SetSignalHandler(INT16 la, NIVXI_HSIGNAL *func);
```

Purpose

Replaces the current signal handler for a logical address with a specified handler.

Parameters

Name	Direction	Description
la	Input	Logical address to set the handler. -1= All known **la** values -2= Unknown **la** (miscellaneous) signal handler
func	Input	Pointer to the new signal handler. Specifying `NULL` will restore the `DefaultSignalHandler`.

Return Values

Return Values	Return Status Description
0	Successful
-1	Invalid **la**

C/C++ Example:

```
/* Set the signal handler for Logical Address 5. */

NIVXI_HSIGNAL  func;
INT16          la;
NIVXI_STATUS   ret;

la = 5;
ret = SetSignalHandler (la, func);
if (ret < 0)
          /* An error occurred in SetSignalHandler. */;


          /* This is a sample VXI signal handler. */
NIVXI_HQUAL void NIVXI_HSPEC func (UINT16 signal)
{
}
```

SetSoftResetHandler

✔ VXI ☐ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_STATUS SetSoftResetHandler(NIVXI_HSOFTRESET *func);
```

Purpose

Replaces the current Soft Reset callback handler with a specified handler.

Parameters

Name	Direction	Description
func	Input	Pointer to the new Soft Reset interrupt callback. Specifying `NULL` will restore the `DefaultSoftResetHandler`.

Return Values

Return Values	Return Status Description
0	Successful
-1	Soft Reset not supported

C/C++ Example:

```
/* Set the Soft Reset handler. */

NIVXI_HSOFTRESET   func;
NIVXI_STATUS       ret;

ret = SetSoftResetHandler (func);
if (ret < 0)
         /* An error occurred in SetSoftResetHandler. */;


/* Example handler */
NIVXI_HQUAL void NIVXI_HSPEC func (void)
{
}
```

SetSysfailHandler

✔ VXI ✔ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_STATUS SetSysfailHandler(NIVXI_HSYSFAIL *func);
```

Purpose

Replaces the current Sysfail callback handler with a specified handler.

Parameters

Name	Direction	Description
func	Input	Pointer to the new Sysfail callback handler. Specifying `NULL` will restore the `DefaultSysfailHandler`.

Return Values

Return Values	Return Status Description
0	Successful
-1	Sysfail not supported

C/C++ Example:

```
/* Set the Sysfail handler. */

NIVXI_HSYSFAIL  func;
NIVXI_STATUS    ret;

ret = SetSysfailHandler (func);
if (ret < 0)
     /* An error occurred in SetSysfailHandler. */;


/* Example handler */
NIVXI_HQUAL void NIVXI_HSPEC func (INT16 controller)
{
}
```

SetSysresetHandler

✔ VXI ✔ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_STATUS SetSysresetHandler(NIVXI_HSYSRESET *func);
```

Purpose

Replaces the current SYSRESET* callback handler with a specified handler.

Parameters

Name	Direction	Description
func	Input	Pointer to the new SYSRESET* callback handler. Specifying `NULL` will restore the `DefaultSysresetHandler`.

Return Values

Return Values	Return Status Description
0	Successful
-1	SYSRESET* not supported

C/C++ Example:

```
/* Set the SYSRESET* handler. */

NIVXI_HSYSRESET  func;
NIVXI_STATUS     ret;

ret = SetSysresetHandler (func);
if (ret < 0)
     /* An error occurred in SetSysresetHandler. */;


/* Example handler */
NIVXI_HQUAL void NIVXI_HSPEC func (INT16 controller)
{
}
```

SetTrigHandler

✔ VXI ☐ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_STATUS SetTrigHandler(UINT16 lines, NIVXI_HTRIG *func);
```

Purpose

Replaces the current TTL/ECL trigger, counter, or tick timer handler for a specified trigger source with the specified function, **func**.

Parameters

Name	Direction	Description
lines	Input	Bit vector of trigger lines (1 = set, 0 = do not set) (See the *Description* section for value descriptions.)
func	Input	Pointer to the new trigger interrupt handler. Specifying `NULL` will restore the `DefaultTrigHandler`. 1 = `DefaultTrigHandler2` Other = Address of new trigger interrupt handler

Return Values

Return Values	Return Status Description
0	Successful
-1	No hardware support

Description

The following table defines the value of the **lines** parameter.

Value	Trigger Line
0 to 7	TTL trigger lines 0 to 7
8 to 13	ECL trigger lines 0 to 5
14	TIC counter*
15	TIC tick timers*
* Only with controllers that have the TIC ASIC	

C/C++ Example:

```
/* Set a trigger handler for TTL trigger line 4. */

NIVXI_HTRIG    func;
UINT16         lines;
NIVXI_STATUS   ret;

lines = (UINT16)(1<<4);
ret = SetTrigHandler (lines, func);
if (ret < 0)
     /* An error occurred in SetTrigHandler . */;


/* Example handler */
NIVXI_HQUAL void NIVXI_HSPEC func (INT16 controller, UINT16 line, UINT16
type)
{
}
```

SetVXIintHandler

✔ VXI ✔ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_STATUS SetVXIintHandler(UINT16 levels, NIVXI_HVXIINT *func);
```

Purpose

Replaces the current VXI/VME interrupt handler for the specified VXI/VMEbus interrupt levels with a specified VXI/VME interrupt handler.

Parameters

Name	Direction	Description
levels	Input	Bit vector of VXI/VME interrupt levels. Bits 6 to 0 correspond to VXI/VME interrupt levels 7 to 1, respectively. 1 = Set 0 = Do not set handler
func	Input	Pointer to the new VXI/VME interrupt handler. Specifying `NULL` will restore the `DefaultVXIintHandler`.

Return Values

Return Values	Return Status Description
0	Successful
-1	No hardware support

C/C++ Example:

```
/* Set the VXI/VME handler for VXI/VME interrupt level 4. */

NIVXI_HVXIINT  func;
UINT16         levels;
NIVXI_STATUS   ret;

levels = (UINT16)(1<<3);
ret = SetVXIintHandler (levels, func);
if (ret < 0)
     /* An error occurred in SetVXIintHandler. */;


 /* This is a sample VXI/VME interrupt handler. */
NIVXI_HQUAL void NIVXI_HSPEC func (INT16 controller, UINT16 level,
UINT32 statusId)
{
}
```

SetWSScmdHandler

☑ VXI ☐ VME ☑ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_STATUS SetWSScmdHandler(NIVXI_HWSSCMD *func);
```

Purpose

Replaces the current WSScmd handler with a specified handler.

Parameters

Name	Direction	Description
func	Input	Pointer to the new WSScmd interrupt handler. Specifying NULL will restore the DefaultWSScmdHandler.

Return Values

Return Values	Return Status Description
0	Successful
-1	Servant Word Serial functions not supported

C/C++ Example:

```
/* Set the WSScmd handler. */

NIVXI_HWSSCMD  func;
NIVXI_STATUS   ret;

ret = SetWSScmdHandler(func);
if (ret < 0)
     /* An error occurred in SetWSScmdHandler. */;


/* Example handler */
NIVXI_HQUAL void NIVXI_HSPEC func (UINT16 cmd)
{
}
```

SetWSSEcmdHandler

✔ VXI ☐ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_STATUS SetWSSEcmdHandler(NIVXI_HWSSECMD *func);
```

Purpose

Replaces the current WSSEcmd handler with a specified handler.

Parameters

Name	Direction	Description
func	Input	Pointer to the new WSSEcmd handler. Specifying NULL will restore the DefaultWSSEcmdHandler.

Return Values

Return Values	Return Status Description
0	Successful
-1	Servant Word Serial functions not supported

C/C++ Example:

```
/* Set the WSSEcmd handler. */

NIVXI_HWSSECMD  func;
NIVXI_STATUS    ret;

ret = SetWSSEcmdHandler(func);
if (ret < 0)
     /* An error occurred in SetWSSEcmdHandler. */;


/* Example handler */
NIVXI_HQUAL void NIVXI_HSPEC func (UINT16 cmdExt, UINT32 cmd)
{
}
```

SetWSSLcmdHandler

✔ VXI ☐ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_STATUS SetWSSLcmdHandler(NIVXI_HWSSLCMD *func);
```

Purpose

Replaces the current WSSLcmd handler with a specified handler.

Parameters

Name	Direction	Description
func	Input	Pointer to the new WSSLcmd handler. Specifying NULL will restore the DefaultWSSLcmdHandler.

Return Values

Return Values	Return Status Description
0	Successful
-1	Servant Word Serial functions not supported

C/C++ Example:

```
/* Set the WSSLcmd handler. */


NIVXI_HWSSLCMD  func;
NIVXI_STATUS    ret;

ret = SetWSSLcmdHandler(func);
if (ret < 0)
     /* An error occurred in SetWSSLcmdHandler. */;


/* Example handler */
NIVXI_HQUAL void NIVXI_HSPEC func (UINT32 cmd)
{
}
```

SetWSSrdHandler

✔ VXI ☐ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_STATUS SetWSSrdHandler(NIVXI_HWSSRD *func);
```

Purpose

Replaces the current WSSrd done notification handler with a specified handler.

Parameters

Name	Direction	Description
func	Input	Pointer to the new WSSrd done notification handler. Specifying NULL will restore the DefaultWSSrdHandler.

Return Values

Return Values	Return Status Description
0	Successful
-1	Servant Word Serial functions not supported

C/C++ Example:

```
/* Set the WSSrd done notification handler. */

NIVXI_HWSSRD    func;
NIVXI_STATUS    ret;

ret = SetWSSrdHandler(func);
if (ret < 0)
          /* An error occurred in SetWSSrdHandler. */;


/* Example handler */
NIVXI_HQUAL void NIVXI_HSPEC func (INT16 status, UINT32 count)
{
}
```

SetWSSwrtHandler

☑ VXI ☐ VME ☑ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_STATUS SetWSSwrtHandler(NIVXI_HWSSWRT *func);
```

Purpose

Replaces the current `WSSwrt` done notification interrupt handler with a specified handler.

Parameters

Name	Direction	Description
func	Input	Pointer to the new `WSSwrt` done notification handler. Specifying `NULL` will restore the `DefaultWSSwrtHandler`.

Return Values

Return Values	Return Status Description
0	Successful
-1	Servant Word Serial functions not supported

C/C++ Example:

```
/* Set the WSSwrt done notification interrupt handler. */

NIVXI_HWSSWRT   func;
NIVXI_STATUS    ret;

ret = SetWSSwrtHandler(func);
if (ret < 0)
         /* An error occurred in SetWSSwrtHandler. */;


/* Example handler */
NIVXI_HQUAL void NIVXI_HSPEC func (INT16 status, UINT32 count)
{
}
```

SignalDeq

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS SignalDeq(INT16 la, UINT32 signalmask, UINT16
*sigval);
```

BASIC Syntax

```
FUNCTION SignalDeq% (BYVAL la%, BYVAL signalmask&, sigval%)
```

Purpose

Gets a signal specified by the signalmask from the signal queue for the specified logical address.

Parameters

Name	Direction	Description
la	Input	Logical address to dequeue signal from. (255 = VME; -1 = any known **la**)
signalmask	Input	A bit vector indicating the type of signal to dequeue; a one in any bit position causes the subroutine to dequeue signals of the associated type (See the *Description* section for more information.)
sigval	Output	Signal value dequeued from the signal queue

Return Values

Return Values	Return Status Description
ret	Return status bit vector (See the *Description* section for more information.) 0 = A signal was returned in **signal** -1 = The signal queue is empty or no match

Description

Bit	Event Signal
If **la** is a message-based device	
14	User-Defined events
13	VXI Reserved events
12	Shared Memory events
11	Unrecognized Command events
10	Request False (REQF) events
9	Request True (REQT) events
8	No Cause Given events

Bit	Response Signal
If **la** is a message-based device	
7	Unused
6	B14 (Reserved for future definition)
5	Data Out Ready (DOR)
4	Data In Ready (DIR)
3	Protocol Error (ERR)
2	Read Ready (RR)
1	Write Ready (WR)
0	Fast Handshake (FHS)

Bit	Type of Signal (Status/ID) Values
If **la** is *not* a message-based device	
15 to 8	Active high bit (if 1 in bits 15 to 8, respectively)
7 to 0	Active low bit (if 0 in bits 15 to 8, respectively)

C/C++ Example:

```
/* Dequeue any type of signal from the signal queue for Logical Address
   10. */

NIVXI_STATUS   ret;
INT16          la;
UINT16         sigval;
UINT32         signalmask;

la = 10;
signalmask = 0xffffL;
ret = SignalDeq (la, signalmask, &sigval);
if (ret < 0)
         /* Empty signal queue for Logical Address 10. */;
```

BASIC Example:

```
' Dequeue any type of signal from the signal queue for Logical
' Address 10.

la% = 10
signalmask& = &HFFFF&
ret% = SignalDeq% (la%, signalmask&, sigval%)
IF ret% < 0 THEN
         ' Empty signal queue for Logical Address 10.
END IF
```

SignalEnq

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS SignalEnq(UINT16 sigval);
```

BASIC Syntax

```
FUNCTION SignalEnq% (BYVAL sigval%)
```

Purpose

Puts a signal on the tail of the signal queue.

Parameters

Name	Direction	Description
sigval	Input	Value to enqueue at the tail of the signal queue

Return Values

Return Values	Return Status Description
0	**sigval** was queued
-1	**sigval** was not queued because the signal queue is full
-2	**sigval** was not queued because the logical address is invalid

C/C++ Example:

```
/* Enqueue signal 0xfd02 (REQT for Logical Address 2) at the tail of the
   signal queue. */

NIVXI_STATUS   ret;
UINT16         sigval;

signal = 0xfd02;
ret = SignalEnq (sigval);
if (ret < 0)
/* Enqueue failed. */;
```

BASIC Example:

```
' Enqueue signal &HFD02 (REQT for Logical Address 2) at the tail
' of the signal queue.

sigval% = &HFD02
ret% = SignalEnq% (sigval%)
IF ret% < 0 THEN
          ' Enqueue failed.
END IF
```

SignalJam

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS SignalJam(UINT16 sigval);
```

BASIC Syntax

```
FUNCTION SignalJam% (sigval%)
```

Purpose

Puts a signal on the head of the signal queue.

☞ **Note:** ***This function is intended only for debugging purposes.***

Parameters

Name	Direction	Description
sigval	Input	Signal value to put on the head of the queue

Return Values

Return Values	Return Status Description
0	**sigval** was queued
-1	**sigval** was not queued because the signal queue is full
-2	**sigval** was not queued because the logical address is invalid

C/C++ Example:

```
/* Put signal 0xfd02 (REQT for Logical Address 2) on the head of the
   signal queue. */

NIVXI_STATUS   ret;
UINT16         sigval;

sigval = 0xfd02;
ret = SignalJam (sigval);
if (ret < 0)
         /* Signal jam failed. */;
```

BASIC Example:

```
' Put signal &HFD02 (REQT for Logical Address 2) on the head of the
  signal queue.

sigval% = &HFD02
ret% = SignalJam% (sigval%)
IF ret% < 0 THEN
         ' signal jam failed.
END IF
```

SrcTrig

✔ VXI ☐ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS SrcTrig(INT16 controller, UINT16 line, UINT16 prot,
INT32 timeout);
```

BASIC Syntax

```
FUNCTION SrcTrig% (BYVAL controller%, BYVAL line%, BYVAL prot%,
BYVAL timeout&)
```

Purpose

Sources the specified protocol on the specified TTL, ECL, or external trigger line on the specified controller.

Parameters

Name	Direction	Description
controller	Input	Controller on which to source trigger line
line	Input	Trigger line to source. (See the *Description* section for value descriptions.)
prot	Input	Protocol to use. (See the *Description* section for value descriptions.)
timeout	Input	Timeout value in milliseconds

Return Values

Return Values	Return Status Description
0	Successful
-1	Unsupported function (no hardware support)
-2	Invalid **controller**
-3	Invalid **line** or **prot**
-4	**line** not supported
-5	**prot** not supported

Return Values	Return Status Description
-6	Timeout occurred waiting for acknowledge
-7	**line** already in use
-12	**line** not configured for use in sourcing
-15	Previous operation incomplete
-16	Previous acknowledge still pending

Description

The following table defines the value of the **line** parameter.

Value	Trigger Line
0 to 7	TTL trigger lines 0 to 7
8 to 13	ECL trigger lines 0 to 5
40 to 49	External source/destination (GPIOs 0 to 9) *
50	TIC counter**
60	TIC tick timers**
* Supports only ON, OFF, START, STOP, and SYNC protocols ** Supports only SYNC and SEMI-SYNC protocols and controllers that have the TIC ASIC.	

The following table defines the value of the **prot** parameter.

Value	Protocol
0	ON
1	OFF
2	START
3	STOP
4	SYNC
5	SEMI-SYNC
6	ASYNC
7	SEMI-SYNC and wait for acknowledge
8	ASYNC and wait for acknowledge
ffffh	Abort previous acknowledge pending (5 and 6)

C/C++ Example:

```
/* Source TTL line 4 on the local CPU (or the first remote controller)
   for SEMI-SYNC protocol. */

NIVXI_STATUS    ret;
INT16           controller;
UINT16          line;
UINT16          prot;
INT32           timeout;

controller = -1;
line = 4;
prot = 5;
timeout = 0L;
ret = SrcTrig (controller, line, prot, timeout);
```

BASIC Example:

```
' Source ECL line 2 on the local CPU (or the first remote
' controller) for SEMI-SYNC protocol.

controller% = -1
line% = 10
prot% = 5
timeout& = 0&
ret% = SrcTrig% (controller%, line%, prot%, timeout&)
```

TrigAssertConfig

✔ VXI ☐ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS TrigAssertConfig(INT16 controller, UINT16 line,
UINT16 configmode);
```

BASIC Syntax

```
FUNCTION TrigAssertConfig% (BYVAL controller%, BYVAL line%, BYVAL
configmode%)
```

Purpose

Configures the specified TTL/ECL trigger line assertion method.

Parameters

Name	Direction	Description
controller	Input	Controller on which to configure assertion mode
line	Input	Trigger line to configure (See the *Description* section for value descriptions.)
configmode	Input	Configuration mode (See the *Description* section for possible bit values.)

Return Values

Return Values	Return Status Description
0	Successful
-1	Unsupported function (no hardware support)
-2	Invalid **controller**
-3	Invalid **line**
-4	**line** not supported
-10	Invalid **configmode**

Description

You can (re-)synchronize TTL/ECL triggers to CLK10 on a per-line basis. You can globally select on all TTL/ECL trigger lines whether to synchronize to the rising or falling edge of CLK10. In addition, you can specify a trigger line to partake in SEMI-SYNC accepting with external acknowledge. The following table defines the value of the **line** parameter.

Value	Trigger Line
0 to 7	TTL trigger lines 0 to 7
8 to 13	ECL trigger lines 0 to 5
ffffh	General assertion configuration (all lines)

The following table defines the value of the **configmode** parameter.

Bit	Description
General Line Configuration Modes (Select all lines)	
0	1 = Synchronize falling edge of CLK10 0 = Synchronize rising edge of CLK10
Specific Configuration Modes (Select individual lines)	
0	1 = Pass trigger through asynchronously 0 = Synchronize with next CLK10 edge
All other values are reserved for future expansion.	
1	1 = Participate in SEMI-SYNC with external trigger acknowledge protocol 0 = Do not participate

C/C++ Example 1:

```
/*  Configure all TTL/ECL trigger lines generally to synchronize to the
    falling edge of CLK10 (as opposed to the rising edge). */

NIVXI_STATUS   ret;
INT16          controller;
UINT16         line;
UINT16         configmode;

controller = -1;
line = 0xFFFF;
configmode = (1<<0);
ret = TrigAssertConfig (controller, line, configmode);
```

C/C++ Example 2:

```
/*  Configure TTL trigger line 4 to synchronize to CLK10 for any
    assertion method and do not participate in SEMI-SYNC. */

NIVXI_STATUS   ret;
INT16          controller;
UINT16         line;
UINT16         configmode;

controller = -1;
line = 4;
configmode = 0;
ret = TrigAssertConfig (controller, line, configmode);
```

BASIC Example 1:

```
' Configure all TTL/ECL trigger lines to synchronize to
' the falling edge of CLK10 (as opposed to the rising edge).

controller% = -1
line% = -1
configmode% = 1
ret% = TrigAssertConfig% (controller%, line%, configmode%)
```

BASIC Example 2:

```
' Configure TTL trigger line 4 to synchronize to CLK10 for any
' assertion method and do not participate in SEMI-SYNC.

controller% = -1
line% = 4
configmode% = 0
ret% = TrigAssertConfig% (controller%, line%, configmode%)
```

TrigCntrConfig

✔ VXI ☐ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS TrigCntrConfig(INT16 controller, UINT16 configmode,
UINT16 source, UINT16 count);
```

BASIC Syntax

```
FUNCTION TrigCntrConfig% (BYVAL controller%, BYVAL configmode%,
BYVAL source%, BYVAL count%)
```

Purpose

Configures TIC chip internal 16-bit counter.

Parameters

Name	Direction	Description
controller	Input	Controller on which to configure the TIC counter
configmode	Input	Configuration mode (See the *Description* section for value descriptions.)
source	Input	Trigger line to configure as input to counter (See the *Description* section for value descriptions.)
count	Input	Number of input pulses to count before terminating

Return Values

Return Values	Return Status Description
0	Successful
-1	Unsupported function (no hardware support)
-2	Invalid **controller**
-3	Invalid **source** line
-10	Invalid **configmode**
-12	Counter not initialized
-15	Previous **count** incomplete

Description

Call `SrcTrig` or `EnableTrigSense` to actually start the counter. The input can be any trigger line, CLK10, or the EXTCLK connection. The counter has two outputs: TCNTR (one 100 ns pulse per input edge) and GCNTR (unasserted until count goes from 1 to 0, then asserted until counter reloaded or reset). You can use `MapTrigToTrig` to map TCNTR to any number of the TTL or ECL trigger lines, and to map GCNTR to any number of the external (GPIO) lines. The following table defines the value of the **configmode** parameter.

Value	Configuration Mode
0	Initialize the counter
2	Reload the counter leaving enabled
3	Disable/abort any count in progress

The following table defines the value of the **source** parameter.

Value	Trigger Line
0 to 7	TTL trigger lines 0 to 7
8 to 13	ECL trigger lines 0 to 5
70	CLK10
71	EXTCLK connection

C/C++ Example:

```
/* Configure the counter to count 25 assertions on TTL trigger line 5
   (the prot parameter when calling EnableTrigSense will determine
   whether the counter accepts SYNC or SEMI-SYNC assertions). */

NIVXI_STATUS   ret;
INT16          controller;
UINT16         configmode;
UINT16         source;
UINT16         count;

controller = -1;
configmode = 0;          /* Initialize the counter */
source = 5;
count = 25;
ret = TrigCntrConfig (controller, configmode, source, count);
```

BASIC Example:

```
' Configure the counter to count 25 assertions on TTL trigger
' line 5 (the prot parameter in EnableTrigSense determines whether the
' counter accepts SYNC or SEMI-SYNC assertions).

controller% = -1
configmode% = 0          ' Initialize the counter
source% = 5
count% = 25
ret% = TrigCntrConfig% (controller%, configmode%, source%, count%)
```

TrigExtConfig

✔ VXI ☐ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS TrigExtConfig(INT16 controller, UINT16 extline,
UINT16 configmode);
```

BASIC Syntax

```
FUNCTION TrigExtConfig% (BYVAL controller%, BYVAL extline%,
BYVAL configmode%)
```

Purpose

Configures the external trigger (GPIO) lines.

Parameters

Name	Direction	Description
controller	Input	Controller on which to configure the external connection
extline	Input	Trigger line to configure (See the *Description* section for value descriptions.)
configmode	Input	Configuration mode (See the *Description* section for possible bit values.)

Return Values

Return Values	Return Status Description
0	Successful
-1	Unsupported function (no hardware support)
-2	Invalid **controller**
-3	Invalid **extline**
-10	Invalid configuration mode

Description

You can feed back the external trigger lines for use in the crosspoint switch output. You can assert the external trigger lines high or low or leave them unconfigured (tristated) for use as a crosspoint switch input. If you do not feed the external input back, you can invert it before mapping it to a trigger line. The following table defines the value of the **extline** parameter.

Value	Trigger Line
40 to 49	External source/destination (GPIOs 0 to 9)
40	Front panel In (connector 1)
41	Front panel Out (connector 2)
42	ECL bypass from front panel
43	EXTCLK
44 to 49	Hardware-dependent GPIOs 4 to 9

The following table defines the value of the **configmode** parameter.

Bit	Configuration Modes
0	1 = Feed back any line mapped as input into the crosspoint switch 0 = Drive input to external (GPIO) pin
1	1 = Assert input (regardless of feedback) 0 = Leave input unconfigured
2	1 = If assertion selected, assert low 0 = If assertion selected, assert high
3	1 = Invert external input (not feedback) 0 = Pass external input unchanged
All other values are reserved for future expansion.	

C/C++ Example 1:

```
/* Configure external line 41 (front panel Out) to not be fed back and
   left tristated for use as a mapped output via MapTrigToTrig. */

NIVXI_STATUS   ret;
INT16          controller;
UINT16         extline;
UINT16         configmode;

controller = -1;
extline = 41;
configmode = 0;
ret = TrigExtConfig (controller, extline, configmode);
```

C/C++ Example 2:

```
/* Configure external line 40 (front panel In) to not be fed back and
   left tristated for use as a mapped input via MapTrigToTrig. Invert
   the front panel In signal. */

NIVXI_STATUS   ret;
INT16          controller;
UINT16         extline;
UINT16         configmode;

controller = -1;
extline = 40;
configmode = (1<<3);
ret = TrigExtConfig (controller, extline, configmode);
```

C/C++ Example 3:

```
/*  Configure external line 48 (GPIO 8) to be fed back for use as a
    crosspoint switch input and output via MapTrigToTrig. */

NIVXI_STATUS   ret;
INT16          controller;
UINT16         extline;
UINT16         configmode;

controller = -1;
extline = 48;
configmode = (1<<0);
ret = TrigExtConfig (controller, extline, configmode);
```

BASIC Example 1:

```
' Configure external line 41 (front panel Out) to not be fed back
' and left tristated for use as a mapped output via MapTrigToTrig.

controller% = -1
extline% = 41
configmode% = 0
ret% = TrigExtConfig% (controller%, extline%, configmode%)
```

BASIC Example 2:

```
' Configure external line 40 (front panel In) to not be fed back
' and left tristated for use as a mapped input via MapTrigToTrig.
' Invert the front panel In signal.

controller% = -1
extline% = 40
configmode% = 8
ret% = TrigExtConfig% (controller%, extline%, configmode%)
```

BASIC Example 3:

```
' Configure external line 48 (GPIO 8) to be fed back for use as a
' crosspoint switch input and output via MapTrigToTrig.

controller% = -1
extline% = 48
configmode% = 8
ret% = TrigExtConfig% (controller%, extline%, configmode%)
```

TrigTickConfig

✔ VXI ☐ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS TrigTickConfig(INT16 controller, UINT16 configmode,
UINT16 source, UINT16 tcount1, UINT16 tcount2);
```

BASIC Syntax

```
FUNCTION TrigTickConfig% (BYVAL controller%, BYVAL configmode%,
BYVAL source%, BYVAL tcount1%, BYVAL tcount2%)
```

Purpose

Configures TIC chip internal dual 5-bit tick timers.

Parameters

Name	Direction	Description
controller	Input	Controller on which to configure the TIC chip dual 5-bit tick timers
configmode	Input	Configuration mode (See the *Description* section for value descriptions.)
source	Input	Trigger line to configure as input to counter (See the *Description* section for value descriptions.)
tcount1	Input	Number of input pulses (as a power of two) to count before asserting TICK1 output and terminating the tick timer if configured for non-rollover mode
tcount2	Input	Number of input pulses (as a power of two) to count before asserting TICK2 output

Return Values

Return Values	Return Status Description
0	Successful initialization of rollover mode
1	Successful initialization of non-rollover mode
2	Successful reload of the tick timers
3	Successful disable of the tick timers
-1	Unsupported function (no hardware support)
-2	Invalid **controller**
-3	Invalid **source** line
-10	Invalid **configmode**
-13	Invalid **tcount1** or **tcount2**
-15	Previous tick configured and enabled

Description

Call `SrcTrig` or `EnableTrigSense` to actually start the tick timers. `SrcTrig` inhibits the TICK1 output from generating tick timer interrupts. `EnableTrigSense` enables the TICK1 output to generate tick timer interrupts. The input can be any external (GPIO) line, CLK10, or the EXTCLK connection. You can map the two tick timer outputs TICK1 and TICK2 to any number of TTL/ECL trigger lines. In addition, you can map the TICK2 output to any number of external (GPIO) lines. The following table defines the value of the **configmode** parameter.

Value	Configuration Mode
0	Initialize the tick timers (rollover mode)
1	Initialize the tick timers (non-rollover mode)
2	Reload the tick timers leaving enabled
3	Disable/abort any count in progress

The following table defines the value of the **source** parameter.

Value	Trigger Line
40 to 49	External source/destination (GPIOs 0 to 9)
70	CLK10
71	EXTCLK connection

C/C++ Example 1:

```
/* Configure the tick timers to interrupt every 6.55 milliseconds by
   dividing down CLK10 as an input. Call EnableTrigSense to start the
   tick timers and enable interrupts. */

NIVXI_STATUS   ret;
INT16          controller;
UINT16         configmode;
UINT16         source;
UINT16         tcount1, tcount2;

controller = -1;
configmode = 0;          /* Initialize with rollover */
source = 70;             /* CLK10 */
tcount1 = 16;            /* Divide down by 65536 (2^16) */
tcount2 = 0;             /* Does not matter */
ret = TrigTickConfig (controller, configmode, source, tcount1, tcount2);
```

C/C++ Example 2:

```
/* Configure the tick timers to output a continuous 9.765 kHz square
   wave on TICK1 output and a 1.25 MHz clock on TICK2 output by dividing
   down CLK10 as an input. Call SrcTrig to start the tick timers. */

NIVXI_STATUS   ret;
INT16          controller;
UINT16         configmode;
UINT16         source;
UINT16         tcount1, tcount2;

controller = -1;
configmode = 0;          /* Initialize with rollover */
source = 70;             /* CLK10 */
tcount1 = 10;            /* Divide down by 1024 (2^10) */
tcount2 = 3;             /* Divide down by 8 (2^3)*/
ret = TrigTickConfig (controller, configmode, source, tcount1, tcount2);
```

BASIC Example 1:

```
' Configure the tick timers to interrupt every 6.55 milliseconds
' by dividing down CLK10 as an input. Call EnableTrigSense to
' start the tick timers and enable interrupts.

controller% = -1
configmode% = 0    ' Initialize with rollover
source% = 70       ' CLK10
tcount1% = 16      ' Divide down by 65536 (2^16)
tcount2% = 0       ' Does not matter
ret% = TrigTickConfig% (controller%, configmode%, source%,
                   tcount1&, tcount2&)
```

BASIC Example 2:

```
' Configure the tick timers to output a continuous 9.765-kHz
' square wave on TICK1 output and a 1.25 MHz clock on TICK2 output
' by dividing down CLK10 as an input. Call SrcTrig to start the
' tick timers.

controller% = -1
configmode% = 0    ' Initialize with rollover
source% = 70       ' CLK10
tcount1& = 10      ' Divide down by 1024 (2^10)
tcount2& = 3       ' Divide down by 8
ret% = TrigTickConfig% (controller%, configmode%, source%,
                   tcount1&, tcount2&)
```

UnMapTrigToTrig

✔ VXI ☐ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS UnMapTrigToTrig(INT16 controller, UINT16 srcTrig,
UINT16 destTrig);
```

BASIC Syntax

```
FUNCTION UnMapTrigToTrig% (BYVAL controller%, BYVAL srcTrig%,
BYVAL destTrig%)
```

Purpose

Unmaps the specified TTL, ECL, Star X, Star Y, external connection (GPIO), or miscellaneous signal line that was mapped to another line using the `MapTrigToTrig` function.

Parameters

Name	Direction	Description
controller	Input	Controller on which to unmap signal lines
srcTrig	Input	Source line to unmap from destination (See the *Description* section for value descriptions.)
destTrig	Input	Destination line mapped from source (See the *Description* section for value descriptions.)

Return Values

Return Values	Return Status Description
0	Successful
-1	Unsupported function, no mapping capability
-2	Invalid **controller**
-12	Not previously mapped

Description

The following table defines the value of the **srcTrig** or **destTrig** parameters.

Value	Source or Destination
0 to 7	TTL trigger lines 0 to 7
8 to 13	ECL trigger lines 0 to 5
14 to 26	Star X lines 0 to 12 *
27 to 39	Star Y lines 0 to 12 *
40 to 49	External source/destination (GPIOs 0 to 9)
40	Front panel In (connector 1)
41	Front panel Out (connector 2)
42	ECL bypass from front panel
43	Connection to EXTCLK input pin
44 to 49	Hardware-dependent GPIOs 4 to 9
50	TIC counter pulse output (TCNTR)
51	TIC counter finished output (GCNTR)
60	TIC TICK1 tick timer output
61	TIC TICK2 tick timer output
* Star X and Star Y are not currently supported lines.	

C/C++ Example:

```
/* Unmap route of TTL line 4 on the local CPU (or first remote
   controller) to go out of the front panel as mapped by
   MapTrigToTrig(). */

INT16          controller;
UINT16         srcTrig;
UINT16         destTrig;
NIVXI_STATUS   ret;

controller = -1;   /* Local CPU. */
src = 4;           /* TTL line 4. */
dest = 49;         /* Front panel out connector. */
ret = UnMapTrigToTrig (controller, srcTrig, destTrig);
```

BASIC Example:

```
' Unmap route of TTL line 4 on the local CPU (or first remote
' controller) to go out of the front panel as mapped by
' MapTrigToTrig.

controller% = -1            ' Local CPU
srcTrig% = 4                ' TTL line 4
destTrig% = 49              ' Front panel out connector
ret% = UnMapTrigToTrig% (controller%, srcTrig%, destTrig%)
```

UnMapVXIAddress

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS UnMapVXIAddress(UINT32 windownum);
```

BASIC Syntax

```
FUNCTION UnMapVXIAddress% (BYVAL windownum&)
```

Purpose

Deallocates a window that was allocated using the `MapVXIAddress` function.

Parameters

Name	Direction	Description
windownum	Input	Window number obtained from `MapVXIAddress`

Return Values

Return Values	Return Status Description
0	window successfully unmapped
1	Access Only released (Window is still mapped due to multiple `MapVXIAddress` calls)
-1	Invalid **windownum**

C/C++ Example:

```
/* Unmap the window obtained from MapVXIAddress. */

UINT16          accessparms;
UINT32          address;
INT32           timo;
UINT32          windownum;
NIVXI_STATUS    ret;
void            *addr;

accessparms = 1;
address = 0xc100L;
timo  = 0L;
addr = MapVXIAddress (accessparms, address, timo, &windownum, &ret);
if (ret >= 0)
{
          /**
          Use the pointer here.
          **/
          ret = UnMapVXIAddress (windownum);
          if (ret < 0)
             /** Unmap unsuccessful. **/
}
```

BASIC Example:

```
' Unmap the window obtained from MapVXIAddress.

accessparms% = 1
address& = &HC100&
timo&  = 0&
addr& = MapVXIAddress& (accessparms%, address&, timo&, windownum&,
                        ret%)
IF ret% >= 0 THEN
   ' Use the address here.
          ret% = UnMapVXIAddress% (windownum&)
          IF ret% < 0 THEN
             ' Unmap unsuccessful.
          END IF
END IF
```

VXIin

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS VXIin(UINT16 accessparms, UINT32 address, UINT32
accwidth, void *value);
```

BASIC Syntax

```
FUNCTION VXIin% (BYVAL accessparms%, BYVAL address&, BYVAL
accwidth%, value AS ANY)
```

Purpose

Reads a single byte, word, or longword from a specified VXI/VME address with the specified byte order and privilege state.

Parameters

Name	Direction	Description
accessparms	Input	See the bit description table in the *Description* section for possible values.
address	Input	VXI/VME address within specified space
accwidth	Input	Read width 1 = Byte 2 = Word 4 = Longword
value	Output	Value read (`UINT8`, `UINT16`, or `UINT32`) (C/C++) Value read (byte, integer, or long) (BASIC)

Return Values

Return Values	Return Status Description
0	Read completed successfully
-1	Bus error occurred during transfer
-2	Invalid parms
-3	Invalid **address**
-4	Invalid **accwidth**
-5	Byte order not supported
-6	**address** not accessible with this hardware
-7	Privilege not supported
-9	**accwidth** not supported

Description

The following table defines the value of the **accessparms** parameter.

Bit	Description
0, 1	VXI/VME Address Space 1 = A16 2 = A24 3 = A32
2 to 4	Access Privilege 0 = Nonprivileged data access 1 = Supervisory data access 2 = Nonprivileged program access 3 = Supervisory program access 4 = Nonprivileged block access 5 = Supervisory block access
5, 6	Reserved (should be 0)
7	Byte Order 0 = Motorola 1 = Intel
8 to 15	Reserved (should be 0)

C/C++ Example:

```
/* Read ID register of the device at Logical Address 4. */

NIVXI_STATUS   ret;
UINT16         accessparms;
UINT32         address;
UINT16         accwidth;
UINT16         value;

accessparms = 1;
address = 0xc100L;
accwidth = 2;
ret = VXIin (accessparms, address, accwidth, &value);
if (ret < 0)
         /* An error occurred during read. */;
```

BASIC Example:

```
' Read Protocol register of the device at Logical
' Address 4.

accessparms% = 1
address& = &HC108&       ' &HC000 + LA *
                         ' &H40 + Protocol register offset 8.
accwidth% = 2
ret% = VXIin% (accessparms%, address&, accwidth%,
               value%)
IF ret% < 0 THEN
         ' Error occurred during read.
END IF
```

VXIinLR

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS VXIinLR(UINT16 reg, UINT16 accwidth, void *value);
```

BASIC Syntax

```
FUNCTION VXIinLR% (BYVAL reg%, BYVAL accwidth%, value AS ANY)
```

Purpose

Reads a single byte, word, or longword from a particular VXI/VME register on the local VXI/VME device.

Parameters

Name	Direction	Description
reg	Input	Offset within VXI/VME logical address registers
accwidth	Input	Byte, word, or longword 1 = Byte 2 = Word 4 = Longword
value	Output	Data value read (`UINT8`, `UINT16`, or `UINT32`) (C/C++) Data value read (byte, integer, or long) (BASIC)

Return Values

Return Values	Return Status Description
0	Successful
-1	Bus error
-3	Invalid **reg**
-4	Invalid **accwidth**
-9	**accwidth** not supported

Description

The register is read in Motorola byte order and as nonprivileged data.

C/C++ Example:

```
/* Read the value of the local Offset register. */

NIVXI_STATUS   ret;
UINT16         reg;
UINT16         accwidth;
UINT16         value;

reg = 6;           /* Register offset within registers. */
accwidth = 2;      /* Read word register. */
ret = VXIinLR (reg, accwidth, &value);
if (ret < 0)
         /* An error occurred in VXIinLR. */;
```

BASIC Example:

```
' Read the value of the local Offset register.

reg% = 6           ' Register offset within registers.
accwidth% = 2      ' Read word register.
ret% = VXIinLR% (reg%, accwidth%, value%)
IF ret% < 0 THEN
        ' Error in VXIinLR.
END IF
```

VXIinReg

✔ VXI ☐ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS VXIinReg(INT16 la, UINT16 reg, UINT16 *value);
```

BASIC Syntax

```
FUNCTION VXIinReg% (BYVAL la%, BYVAL reg%, value%)
```

Purpose

Reads a single word from a specified VXI register offset on the specified VXI device.

Parameters

Name	Direction	Description
la	Input	Logical address of device to read from
reg	Input	Offset within VXI logical address registers
value	Output	Value read from device's VXI register

Return Values

Return Values	Return Status Description
0	Read completed successfully
-1	Bus error occurred during transfer
-3	Invalid **reg** specified

Description

The register is read in Motorola byte order and as nonprivileged data. Notice that this operation allows access to the upper 16 KB of A16 space using the logical address scheme defined by the VXI specification. The scheme is as follows:

```
address = 0xC000 + (la * 0x40)
```

where **la** is the logical address and **address** is the resulting A16 space address.

C/C++ Example:

```
/* Read ID register of the device at Logical Address 4. */

NIVXI_STATUS   ret;
INT16          la;
UINT16         reg;
UINT16         value;

la = 4;
reg = 0;
ret = VXIinReg (la, reg, &value);
if (ret < 0)
         /* An error occurred during read. */;
```

BASIC Example:

```
' Read Protocol register of the device at Logical Address 4.

la% = 4
reg% = 8    ' Protocol register offset.
ret% = VXIinReg% (la%, reg%, value%)
IF ret% < 0 THEN
         ' Error occurred during read.
END IF
```

VXIintAcknowledgeMode

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS VXIintAcknowledgeMode(INT16 controller, UINT16
modes);
```

BASIC Syntax

```
FUNCTION VXIintAcknowledgeMode% (BYVAL controller%, BYVAL modes%)
```

Purpose

Specifies whether the VXI/VME interrupt acknowledge cycle for the specified controller for the specified levels should be handled as Release On AcKnowledge (ROAK) interrupts or as Release On Register Access (RORA) interrupts.

Parameters

Name	Direction	Description
controller	Input	Controller on which to specify interrupt acknowledge
modes	Input	Vector of VXI/VMEinterrupt levels to set to RORA/ROAK interrupt acknowledge mode. Bits 6 to 0 correspond to VXI/VME interrupt levels 7 to 1, respectively 0 = Set to ROAK VXI/VME interrupt for corresponding level 1 = Set to RORA VXI/VME interrupt for corresponding level

Return Values

Return Values	Return Status Description
0	Successful VXI/VME interrupt enabled
-1	No hardware support
-2	Invalid **controller**
-5	Invalid **modes** specified

Description

If the VXI/VME interrupt level is handled as a RORA VXI/VME interrupt, the local interrupt generation is automatically inhibited when the VXI/VME interrupt acknowledge is performed. `EnableVXIint` or `EnableVXItoSignalInt` must be called to re-enable the appropriate VXI/VME interrupt level whenever a RORA VXI/VME interrupt occurs.

C/C++ Example:

```
/* Set VXI/VME Interrupt levels 2 and 3 on the local CPU (or first
   extended controller) to be RORA interrupters--set reset to ROAK. */

INT16          controller;
UINT16         modes;
NIVXI_STATUS   ret;

controller = -1;        /** Local CPU or first frame. **/
                        /** Levels 2 and 3 are RORA mode. **/
modes = (UINT16)((1<<1) | (1<<2));
ret = VXIintAcknowledgeMode (controller, modes);
```

BASIC Example:

```
' Set VXI Interrupt levels 2 and 3 on the local CPU (or first
' extended controller) to be RORA interrupters--set reset to ROAK.

controller% = -1   ' Local CPU or first frame.
modes% = &H0006    ' Levels 2 and 3 are RORA mode.
ret% = VXIintAcknowledgeMode% (controller%, modes%)
```

VXImemAlloc

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS VXImemAlloc(UINT32 size, void **useraddr,
UINT32 *vxiaddr);
```

BASIC Syntax

```
FUNCTION VXImemAlloc% (BYVAL size&, useraddr&, vxiaddr&)
```

Purpose

Dynamically allocates a buffer in the dual-ported memory and returns local and VXI/VME addresses.

Parameters

Name	Direction	Description
size	Input	Number of bytes to allocate
useraddr	Output	Returned application memory buffer address
vxiaddr	Output	Returned VXI/VME memory buffer address

Return Values

Return Values	Return Status Description
0	Successful; memory can be accessed directly
1	Successful; memory must be accessed using `VXImemCopy`
-1	Memory allocation failed
-2	Local CPU is A16 only

Description

This function allocates memory that is accessible both through application's memory and VXI/VME address space. The VXI/VME address space is the same as the space for which the local device is dual porting memory. You can use this function for setting up shared memory transfers.

☞ **Note:** ***On some systems you may be required to configure a shared memory pool for*** `VXImemAlloc` ***allocations.***

C/C++ Example:

```
/* Allocate, use, and free 32 kilobytes of VXI/VME Shared system RAM. */

UINT32          size;
void            *useraddr;
UINT32          vxiaddr;
NIVXI_STATUS    ret;

size = 0x8000;     /* 32 kilobytes*/
ret = VXImemAlloc (size, &useraddr, &vxiaddr);
if (ret < 0)
          /* An error occurred in VXImemAlloc. */;

/*
          Use buffer.
*/

ret = VXImemFree (useraddr);
if (ret < 0)
          /* An error occurred in VXImemFree. */;
```

BASIC Example:

```
' Allocate, use, and free 32 kilobytes of VXI/VME Shared system RAM.

Size& = &H8000&    ' 32 kilobytes
ret% = VXImemAlloc% (size&, useraddr&, vxiaddr&)
IF ret% < 0 THEN
          ' Error in VXImemAlloc.
END IF

' Use buffer.

Ret% = VXImemFree% (useraddr&)
IF ret% < 0 THEN
          ' Error in VXImemFree.
END IF
```

VXImemCopy

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS VXImemCopy(void *useraddr, void *bufaddr, UINT32
size, UINT16 dir);
```

BASIC Syntax

```
FUNCTION VXImemCopy% (BYVAL useraddr&, BYVAL bufaddr$, BYVAL
size&, BYVAL dir%)
```

Purpose

Copies an application buffer to or from the local shared memory.

Parameters

Name	Direction	Description
useraddr	Input/Output	User address returned by `VXImemAlloc`
bufaddr	Input/Output	Address of application buffer to copy into or out of
size	Input	Number of bytes to copy
dir	Input	Direction of transfer 1 = Copy from **bufaddr** to **useraddr** 0 = Copy from **useraddr** to **bufaddr**

Return Values

Return Values	Return Status Description
0	Successful
-1	Copy failed
-2	Local CPU is A16 only
-5	Invalid **dir**

Description

On some systems, local shared memory allocated with VXImemAlloc cannot be accessed directly by an application. VXImemCopy provides a fast access method to this memory.

C/C++ Example:

```
/* Allocate, copy, use, and free 32 kilobytes of VXI Shared system RAM. */

UINT32          size;
void            *useraddr;
UINT32          *vxiaddr;
NIVXI_STATUS    ret;
void            *bufaddr;

size = 0x8000;     /* 32 kilobytes. */
ret = VXImemAlloc (size, &useraddr, &vxiaddr);
if (ret < 0)
          /* An error occurred in VXImemAlloc. */;

/*
          Tell remote bus master to copy 32 kilobytes
          to local shared memory by writing to VXI
          address "vxiaddr."
*/

          /* Copy to application. */
bufaddr = malloc(size);
VXImemCopy (useraddr, bufaddr, size, 0);

/*
          Use buffer.
*/

ret = VXImemFree (useraddr);
if (ret < 0)
          /* An error occurred in VXImemFree. */;
```

BASIC Example:

```
' Allocate, copy, use, and free 32 kilobytes of VXI Shared system
' RAM.

DIM bufaddr% (32768)
size& = &H8000&    ' 32 kilobytes
ret% = VXImemAlloc% (size&, useraddr&, vxiaddr&)
IF ret% < 0 THEN
          ' Error in VXImemAlloc.
END IF

' Remote Bus Master access.

IF ret% = 1 THEN
 ret% = VXImemCopy% (useraddr&, bufaddr&, size, 0)
END IF
' Use the buffer.
ret% = VXImemFree% (useraddr&)
```

VXImemFree

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS VXImemFree(void *useraddr);
```

BASIC Syntax

```
FUNCTION VXImemFree% (BYVAL useraddr&)
```

Purpose

Deallocates shared memory buffer that was allocated using the `VXImemAlloc` function.

Parameters

Name	Direction	Description
useraddr	Input	Application memory buffer address to free

Return Values

Return Values	Return Status Description
0	Successful
-1	Memory deallocation failed

C/C++ Example:

```
/* Allocate, use, and free 32 kilobytes of Shared system RAM. */

UINT32          size;
void            *useraddr;
UINT32          vxiaddr;
NIVXI_STATUS    ret;

size = 0x8000;     /* 32 kilobytes. */
ret = VXImemAlloc (size, &useraddr, &vxiaddr);
if (ret < 0)
          /* An error occurred in VXImemAlloc. */;

/*
          Use buffer.
*/

ret = VXImemFree (useraddr);
if (ret < 0)
          /* An error occurred in VXImemFree. */;
```

BASIC Example:

```
' Allocate, use, and free 32 kilobytes of VXI Shared system RAM.

size& = &H8000&
ret% = VXImemAlloc% (size&, useraddr&, vxiaddr&)
IF ret% < 0 THEN
          ' Error in VXImemAlloc.
END IF

' Use buffer.

ret% = VXImemFree% (useraddr&)
IF ret% < 0 THEN
          ' Error in VXImemFree.
END IF
```

VXImove

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS VXImove(UINT16 srcparms, UINT32 srcaddr, UINT16
destparms, UINT32 destaddr, UINT32 length, UINT16 accwidth);
```

BASIC Syntax

```
FUNCTION VXImove% (BYVAL srcparms%, BYVAL srcaddr&, BYVAL
destparms%, BYVAL destaddr&, BYVAL length&, BYVAL accwidth%)
```

Purpose

Copies a block of memory from a specified source location in any address space (local, A16, A24, A32) to a specified destination in any address space.

Parameters

Name	Direction	Description
srcparms	Input	See the bit description table in the *Description* section for possible values
srcaddr	Input	Address within source address space. This address is a long integer value if it represents a VXI/VME space or an array address for a local address space.
destparms	Input	See the bit description table in the *Description* section for possible values
destaddr	Input/ Output	Address within destination address space. This address is a long integer value if it represents a VXI/VME space or an array address for a local address space.
length	Input	Number of elements to transfer
accwidth	Input	Byte, word, or longword 1 = Byte 2 = Word 4 = Longword

Return Values

Return Values	Return Status Description
0	Transfer completed successfully
-1	Bus error occurred
-2	Invalid **srcparms** or **destparms**
-3	Invalid **srcaddr** or **destaddr**
-4	Invalid **accwidth**
-5	Byte order not supported
-6	Address not accessible with this hardware
-7	Privilege not supported
-8	Timeout, DMA aborted (if applicable)
-9	**accwidth** not supported

Description

The following table lists the meaning of different bits in **srcparms** and **destparms** arguments. Not all bits are supported on every platform. If there is no hardware support, the bit value will be ignored.

Bit	Description
0, 1	Source Address Space 0 = Local (bits 2, 3, 4, and 7 should be 0) 1 = A16 2 = A24 3 = A32
2 to 4	Access Privilege 0 = Nonprivileged data access 1 = Supervisory data access 2 = Nonprivileged program access 3 = Supervisory program access 4 = Nonprivileged block access 5 = Supervisory block access 6 = VME64 nonprivileged block access 7 = VME64 privileged block access

Bit	Description
5, 6	Reserved (should be 0)
7	Byte Order 0 = Motorola 1 = Intel
8 to 10	Reserved (should be 0)
11	Use programmed I/O (PIO) instead of DMA
12	No increment (FIFO)
13	Disable MXI block mode
14	Disable MXI synchronous mode
15	Reserved (should be 0)

C/C++ Example:

```
/* Move 1 kilobyte from A24 space at 0x200000 to a local buffer. */

NIVXI_STATUS    ret;
UINT16          srcparms;
UINT32          srcaddr;
UINT16          destparms;
UINT32          destaddr;
UINT32          length;
UINT16          accwidth;

srcparms = 2;                   /* A24, nonprivileged data, Motorola */
srcaddr = 0x200000L;
destparms = 0;                  /* Local space. */
length = 0x200L;                /* 512 elements. */
accwidth = 2;                   /* Transfer as words. */
destaddr = (UINT32)malloc(length * accwidth);
                                /* Allocate local buffer. */
ret = VXImove (srcparms, srcaddr, destparms, destaddr, length, accwidth);
if (ret < 0)
          /* An error occurred during VXImove. */;
```

BASIC Example:

```
' Move 1 kilobyte from A24 space at &H200000& to a local buffer.

DIM destaddr AS STRING * 1024
srcparms% = 2      ' A24, nonprivileged data, Motorola.
srcaddr& = &H200000&
destparms% = 0           ' Local space.
length& = &H400&         ' 1 kilobyte.
accwidth% = 2            ' Transfer as words.
ret% = VXImove (srcparms%, srcaddr&, destparms%,
 destaddr$, length%, accwidth%)
IF ret% < 0 THEN
          ' Error occurred during VXImove.
END IF
```

VXIout

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS VXIout(UINT16 accessparms, UINT32 address, UINT16
accwidth, UINT32 value);
```

BASIC Syntax

```
FUNCTION VXIout% (BYVAL accessparms%, BYVAL address&,
BYVAL accwidth%, BYVAL value&)
```

Purpose

Writes a single byte, word, or longword to a specified VXI/VME address with the specified byte order and privilege state.

Parameters

Name	Direction	Description
accessparms	Input	See the bit description table in the *Description* section for possible values
address	Input	VXI/VME address within specified address space
accwidth	Input	Byte, word, or longword 1 = Byte 2 = Word 4 = Longword
value	Input	Data value to write

Return Values

Return Values	Return Status Description
0	Write completed successfully
-1	Bus error occurred during transfer
-2	Invalid **accessparms**
-3	Invalid **address**
-4	Invalid **accwidth**
-5	Byte order not supported
-6	Address not accessible with this hardware
-7	Privilege not supported
-9	**accwidth** not supported

Description

The following table lists the meaning of different bits in the **accessparms** parameter.

Bit	Description
0, 1	VXI/VME Address Space 1 = A16 2 = A24 3 = A32
2 to 4	Access Privilege 0 = Nonprivileged data access 1 = Supervisory data access 2 = Nonprivileged program access 3 = Supervisory program access 4 = Nonprivileged block access 5 = Supervisory block access

Bit	Description
5, 6	Reserved (should be 0)
7	Byte Order 1 = Motorola 2 = Intel
8 to 15	Reserved (should be 0)

C/C++ Example:

```
/* Write the value 0x2000 to the Offset register of the device at
   Logical Address 4. */

NIVXI_STATUS   ret;
UINT16         accessparms;
UINT32         address;
UINT16         accwidth;
UINT32         value;

accessparms = 1;
address = 0xc10aL;
accwidth = 2;
value = 0x2000L;
ret = VXIout (accessparms, address, accwidth, value);
if (ret < 0)
         /* An error occurred during write. */;
```

BASIC Example:

```
' Write the value &HFD04 (the REQT event for Logical Address 4)
' to the Signal register of the device at Logical device at
' Address 0.

accessparms% = 1
address& = &HC008& ' address = &HC000 + LA *&H40 + register offset 8
accwidth% = 2
value& = &HFD04&          ' REQT
ret% = VXIout% (accessparms%, address&, accwidth%, value&)
IF ret% < 0 THEN
         ' Error occurred during write.
END IF
```

VXIoutLR

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS VXIoutLR(UINT16 reg, UINT16 accwidth, UINT32 value);
```

BASIC Syntax

```
FUNCTION VXIoutLR% (BYVAL reg%, BYVAL accwidth%, BYVAL value&)
```

Purpose

Writes a single byte, word, or longword to a particular VXI/VME register on the local VXI/VME device.

Parameters

Name	Direction	Description
reg	Input	Offset within VXI/VME logical address registers
accwidth	Input	Byte, word, or longword 1 = Byte 2 = Word 4 = Longword
value	Input	Data value to write

Return Values

Return Values	Return Status Description
0	Successful
-1	Bus error
-3	Invalid reg
-4	Invalid **accwidth**
-9	**accwidth** not supported

Description

The VXI/VME register on the local VXI/VME device register is written in Motorola byte order and as nonprivileged data.

C/C++ Example:

```
/* Write the value of 0xfd00 (REQT) to the local Signal register. */

NIVXI_STATUS   ret;
UINT16         reg;
UINT16         accwidth;
UINT32         value;

reg = 8;              /* Register offset for Signal register. */
accwidth = 2;         /* Word register. */
value = 0xfd00L;
ret = VXIoutLR (reg, accwidth, value);
if (ret < 0)
         /* An error occurred in VXIoutLR. */;
```

BASIC Example:

```
' Write the value of &HFD00 (REQT) to the local Signal register.

reg% = 8           ' Register offset for Signal register.
accwidth% = 2      ' Word register.
value& = &HFD00    ' REQT.
ret% = VXIoutLR% (reg%, accwidth%, value&)
IF ret% < 0 THEN
         ' Error in VXIoutLR.
END IF
```

VXIoutReg

✔ VXI ☐ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS VXIoutReg(INT16 la, UINT16 reg, UINT16 value);
```

BASIC Syntax

```
FUNCTION VXIoutReg% (BYVAL la%, BYVAL reg%, BYVAL value%)
```

Purpose

Writes a single word to a specified VXI register offset on the specified VXI device.

Parameters

Name	Direction	Description
la	Input	Logical address of the device to write to
reg	Input	Offset within VXI logical address registers
value	Input	Value written to device's VXI register

Return Values

Return Values	Return Status Description
0	Write completed successfully
-1	Bus error occurred during transfer
-3	Invalid **reg** specified

Description

The register is written in Motorola byte ordering and as nonprivileged data. Notice that this operation allows access to the upper 16 KB of A16 space using the logical address scheme defined by the VXI specification. The scheme is as follows:

```
address = 0xC000 + (la * 0x40)
```

where **la** is the logical address and **address** is the resulting A16 space address

C/C++ Example:

```
/* Write Signal register of the device at Logical Address 10 with the
   value 0xfd0a (REQT). */

NIVXI_STATUS   ret;
UINT16         la;
UINT16         reg;
UINT16         value;

la = 10;
reg = 8;
value = 0xfd0a;
ret = VXIoutReg (la, reg, value);
if (ret < 0)
         /* An error occurred during write. */;
```

BASIC Example:

```
' Write Signal register of the device at Logical Address 0 with
' the value &HFD0A (REQT for Logical Address 10).

la% = 0
reg% = 8                  ' Signal register offset
value% = &HFD0A           ' REQT for Logical Address 10
ret% = VXIoutReg% (la%, reg%, value%)
IF ret% < 0 THEN
         ' Error occurred during write.
END IF
```

VXIpeek

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
void VXIpeek(void *addressptr, UINT16 accwidth, void *value);
```

BASIC Syntax

```
SUB VXIpeek (BYVAL addressptr&, BYVAL accwidth%, value AS ANY)
```

Purpose

Reads a single byte, word, or longword from a specified VXI/VME address by dereferencing a pointer obtained from `MapVXIAddress`.

Parameters

Name	Direction	Description
addressptr	Input	Address pointer obtained from `MapVXIAddress`
accwidth	Input	Byte, word, or longword 1 = Byte 2 = Word 4 = Longword
value	Output	Data value read (`UINT8`, `UINT16`, or `UINT32`)

Return Values

None

C/C++ Example:

```
/* Read the value from the Offset register of the device at Logical
   Address 4. */

UINT16          accessparms;
UINT32          windownum;
NIVXI_STATUS    ret;
UINT16          *addressptr;
UINT16          value;

accessparms = 1;
addressptr =
          (UINT16 *)MapVXIAddress(accessparms,(UINT32)0xc106,
          (INT32)0x7fffffff, &windownum, &ret);
if (ret >= 0) /* If a valid pointer was returned. */
{
          VXIpeek (addressptr, 2, &value);
}
```

BASIC Example:

```
' Read the value from the VXI Status register of the device at
' Logical Address 4 into value, an integer variable.

accessparms% = 1   ' A16, Motorola, nonprivileged data.
addressptr& = MapVXIAddress (accessparms%, &HC106&, &H7FFFFFFF&,
            windownum&, ret%)
IF ret% >= 0 THEN ' If a valid address was returned.
          CALL VXIpeek (addressptr&, 2, value%)
END IF
```

VXIpoke

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
void VXIpoke(void *addressptr, UINT16 accwidth, UINT32 value);
```

BASIC Syntax

```
SUB VXIpoke (BYVAL addressptr&, BYVAL accwidth%, BYVAL value&)
```

Purpose

Writes a single byte, word, or longword to a specified VXI/VME address by dereferencing a pointer obtained from `MapVXIAddress`.

Parameters

Name	Direction	Description
addressptr	Input	Address pointer obtained from `MapVXIAddress`
accwidth	Input	Byte, word, or longword 1 = Byte 2 = Word 4 = Longword
value	Input	Data value to write

Return Values

None

C/C++ Example:

```
/* Write the value 0x2000 to the Offset register of the device at
   Logical Address 4. */

UINT16          accessparms;
UINT32          windownum;
NIVXI_STATUS    ret;
UINT16          *addressptr;
UINT32          value;

accessparms = 1;
addressptr =
          (UINT16 *)MapVXIAddress(accessparms,(UINT32)0xc106,
          (INT32)0x7fffffff, &windownum, &ret);
if (ret >= 0) /* If a valid pointer was returned. */
{
          value = 0x2000L;
          VXIpoke (addressptr, 2, value);
}
```

BASIC Example:

```
' Write the value &HFD04& (REQT event) to the Signal register of the
  device at Logical Address 0.

accessparms% = 1   ' A16, Motorola, nonprivileged data.
addressptr& = MapVXIAddress (accessparms%, &HC008&, &H7FFFFFFF&,
            windownum&, ret%)
IF ret% >= 0& THEN ' If a valid address was returned.
          value& = &HFD04&
          CALL VXIpoke (addressptr&, 2, value&)
END IF
```

WaitForSignal

✔ VXI ✔ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS WaitForSignal (INT16 la, UINT32 signalmask, INT32
timeout, UINT16 *retsignal, UINT32 *retsignalmask);
```

BASIC Syntax

```
FUNCTION WaitForSignal% (BYVAL la%, BYVAL signalmask&, BYVAL
timeout&, retsignal%, retsignalmask&)
```

Purpose

Waits for a specified type(s) of signal or status/ID to be received from a specified logical address.

Parameters

Name	Direction	Description
la	Input	Logical address of device sourcing the signal (255 = VME; -1 = any known **la**)
signalmask	Input	A bit vector indicating the type(s) of signals that the application will wait for; a one in any bit position will cause the subroutine to detect signals of the associated type (See the *Description* section for more information)
timeout	Input	Time to wait until signal occurs
retsignal	Output	Signal received
retsignalmask	Output	A bit vector indicating the type(s) of signals that the application received. The bits have the same meaning as that of the input **signalmask**.

Return Values

Return Values	Return Status Description
0	One of the specified signals was received
-1	Invalid **la**
-2	Timeout occurred while waiting for the specified signal(s)

Description

The signal you are waiting for must be routed to the signal queue (using the `RouteSignal()` function). The following table lists the meaning of each bit in the **signalmask** and **retsignalmask** arguments.

Bit	Event Signal
If **la** is a message-based device	
14	User-Defined events
13	VXI Reserved events
12	Shared Memory events
11	Unrecognized Command events
10	Request False (REQF) events
9	Request True (REQT) events
8	No Cause Given events

Bit	Response Signal
If **la** is a message-based device	
7	Unused
6	B14 (Reserved for future definition)
5	Data Out Ready (DOR)
4	Data In Ready (DIR)
3	Protocol Error (ERR)
2	Read Ready (RR)
1	Write Ready (WR)
0	Fast Handshake (FHS)

Bit	Type of Signal (Status/ID) Values
If **la** is *not* a message-based device	
15 to 8	Active high bit (if 1 in bits 15 to 8, respectively)
7 to 0	Active low bit (if 0 in bits 15 to 8, respectively)

C/C++ Example:

```
/* Wait 2 seconds for REQT signal from Logical Address 5. */

NIVXI_STATUS    ret;
INT16           la;
UINT32          signalmask;
INT32           timeout;
UINT16          retsignal;
UINT32          retsignalmask;

la = 5;
signalmask = 0x0200L;
timeout = 2000L;
ret = WaitForSignal (la, signalmask, timeout, &retsignal,
     &retsignalmask);
if (ret == 0)
         /* Signal received within specified waiting period. */;
```

BASIC Example:

```
' Wait 2 seconds for REQT signal from Logical Address 5.

la% = 5
signalmask& = &H0200&
timeout& = 2000&   ' 2000 milliseconds = 2 seconds.
ret% = WaitForSignal% (la%, signalmask&, timeout&, retsignal%,
     retsignalmask&)
IF ret% < 0 THEN
          ' Error has occurred.
END IF
```

WaitForTrig

✔ VXI ☐ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS WaitForTrig(INT16 controller, UINT16 line, INT32
timeout);
```

BASIC Syntax

```
FUNCTION WaitForTrig% (BYVAL controller%, BYVAL line%,
BYVAL timeout&)
```

Purpose

Waits for the specified trigger line to be sensed on the specified controller for the specified time.

Parameters

Name	Direction	Description
controller	Input	Controller on which to wait for trigger
line	Input	Trigger line for which to wait on (See the *Description* section for value descriptions)
timeout	Input	Timeout value in milliseconds

Return Values

Return Values	Return Status Description
0	Successful
-1	Unsupported function (no hardware support)
-2	Invalid **controller**
-3	Invalid **line**
-4	**line** not supported
-6	Timeout occurred
-12	**line** not configured for sensing

Description

`EnableTrigSense` must be called to sensitize the hardware to the particular trigger protocol to be sensed. This table lists possible values for the **line** parameter.

Value	Trigger Line
0 to 7	TTL trigger lines 0 to 7
8 to 13	ECL trigger lines 0 to 5
50	TIC counter
60	TIC TICK1 tick timer

C/C++ Example:

```
/* Wait up to 10 seconds for TTL line 4 on the local CPU (or the first
   extended controller) to be encountered. */

NIVXI_STATUS   ret;
INT16          controller;
UINT16         line;
INT32          timeout;

controller = -1;
line = 4;
timeout = 10000L;
ret = WaitForTrig (controller, line, timeout);
```

BASIC Example:

```
' Wait for ECL line 2 on the local CPU (or the first extended
' controller) to be encountered.

controller% = -1
line% = 10
timeout& = 10000&
ret% = WaitForTrig% (controller%, line%, timeout&)
```

WSabort

✔ VXI ☐ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS WSabort (INT16 la, UINT16 abortop);
```

BASIC Syntax

```
FUNCTION WSabort% (BYVAL la%, BYVAL abortop%)
```

Purpose

Performs a Forced or Unrecognized (Unsupported) Command abort of the Commander Word Serial operation(s) in progress.

Parameters

Name	Direction	Description
la	Input	Logical address of the message-based device
abortop	Input	The operation to abort (See the *Description* section for value descriptions)

Return Values

Return Status	Description
0	Successfully aborted
-1	Invalid **la**
-2	Invalid **abortop**

Description

This table lists possible values for the **abortop** parameter.

Value	Description
1	Forced Abort: aborts `WSwrt`, `WSrd`, and `WStrg`
2	UnSupCom: aborts `WScmd`, `WSLcmd`, and `WSEcmd`
3	Forced Abort: aborts `WScmd`, `WSLcmd`, and `WSEcmd`
4	Forced Abort: aborts all Word Serial operations
5	Async Abort: aborts all Word Serial operations immediately. Be careful when using this option. During a Word Serial query, the Servant may be left in an invalid state if the operation is aborted after writing the query and before reading the response register. When using this option, the Word Serial operation aborts immediately unlike options 1, 3, and 4, where the operation does not abort until reading the response.

C/C++ Example:

```
/* Perform Unsupported Command abort on Logical Address 5. */

NIVXI_STATUS    ret;
INT16           la;
UINT16          abortop;

la = 5;
abortop = 2;
ret = WSabort (la, abortop);
if (ret < 0)
          /* An error occurred during WSabort. */;
```

BASIC Example:

```
' Perform Unsupported Command abort on Logical Address 5.

la% = 5
abortop% = 2
ret% = WSabort% (la%, abortop%)
IF ret% < 0 THEN
          ' An error occurred during WSabort.
END IF
```

WSclr

☑ VXI ☐ VME ☑ C/C++ ☑ BASIC

C/C++ Syntax

```
NIVXI_STATUS WSclr (INT16 la);
```

BASIC Syntax

```
FUNCTION WSclr% (BYVAL la%)
```

Purpose

Sends the Word Serial *Clear* command to a message-based device.

Parameters

Name	Direction	Description
la	Input	Logical address of the message-based device

Return Values

Return Values	Return Status Description
ret	Return status bit vector (See the *Description* section for bit descriptions)

Description

The next table gives the meaning of each bit in the return value.

Bit	Name	Description
Bit 15 = 1 (Error Conditions)		
7	BERR	Bus error occurred during transfer
5	InvalidLA	Invalid **la** specified
2	TIMO_DONE	Timed out before WR set (clear complete)
1	TIMO_SEND	Timed out before able to send *Clear*
Bit 15 = 0 (Successful Transfer)		
0	IODONE	Command transfer successfully completed

C/C++ Example:

```
/* Send Clear command to Logical Address 5. */

NIVXI_STATUS   ret;
INT16          la;

la = 5;
ret = WSclr (la);
if (ret < 0)
         /* An error occurred during the command transfer. */;
```

BASIC Example:

```
' Send Clear command to Logical Address 5.

la% = 5
ret% = WSclr% (la%)
IF ret% < 0 THEN
         ' An error occurred during the command transfer.
END IF
```

WScmd / WSEcmd / WSLcmd

✔ VXI ☐ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS WScmd (INT16 la, UINT16 cmd, UINT16 respflag, UINT16
*response);

NIVXI_STATUS WSEcmd (INT16 la, UINT16 cmdExt, UINT32 cmd, UINT16
respflag, UINT32 *response);

NIVXI_STATUS WSLcmd (INT16 la, UINT32 cmd, UINT16 respflag, UINT32
*response);
```

BASIC Syntax

```
FUNCTION WScmd% (BYVAL la%, BYVAL cmd%, BYVAL respflag%,
response%)

FUNCTION WSEcmd% (BYVAL la%, BYVAL cmdExt%, BYVAL cmd&,
BYVAL respflag%, response&)

FUNCTION WSLcmd% (BYVAL la%, BYVAL cmd&, BYVAL respflag%,
response&)
```

Purpose

Sends a Word Serial, Extended Longword Serial, or Longword Serial command or query to a message-based device.

Parameters

Name	Direction	Description
la	Input	Logical address of the message-based device
cmd	Input	Word Serial command value
respflag	Input	Non-0 = Get a response (query) 0 = Do not get a response
response	Output	16 or 32 bit location to store response

Return Values

Return Values	Return Status Description
ret	Return status bit vector

Description

The next table gives the meaning of each bit in the return value.

Bit	Name	Description
Bit 15 = 1 (Error Conditions)		
14	WRviol	Write Ready protocol violation during transfer
13	RRviol	Read Ready protocol violation during transfer
12	DORviol	Data Out Ready protocol violation
11	DIRviol	Data In Ready protocol violation
10	RdProtErr	Read protocol error
9	UnSupCom	Device did not recognize the command
7	BERR	Bus error occurred during transfer
6	MQE	Multiple query error occurred during transfer
5	InvalidLA	Invalid **la** specified
2	TIMO_RES	Timed out before response received
1	TIMO_SEND	Timed out before able to send command
Bit 15 = 0 (Successful Transfer)		
0	IODONE	Command transfer successfully completed

C/C++ Example:

```
/* Send the Word Serial command Read STB to a device at Logical Address
   5 and get the response. */

 NIVXI_STATUS   ret;
 INT16          la;
 UINT16         cmd;
 UINT16         respflag;
 UINT16         response;

 la = 5;
 cmd = 0xcfff;
 respflag = 1;
 ret = WScmd (la, cmd, respflag, &response);
 if ( ret < 0)
          /* An error occurred during WS command transfer. */;
```

BASIC Example:

```
' Send the Word Serial command Read STB to a device at Logical
' Address 5 and get the response.

la% = 5
cmd% = &HCFFF
respflag% = 1
ret% = WScmd% (la%, cmd%, respflag%, response%)
IF ret% < 0 THEN
          ' Error occurred during WS command transfer.
END IF
```

WSgetTmo

✔ VXI ☐ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS WSgetTmo(UINT32 *actualtimo);
```

BASIC Syntax

```
FUNCTION WSgetTmo% (actualtimo&)
```

Purpose

Gets the actual time period to wait before aborting a Word Serial, Longword Serial, or Extended Longword Serial Protocol transfer.

Parameters

Name	Direction	Description
actualtimo	Output	Timeout period in milliseconds

Return Values

Return Values	Return Status Description
0	Successful

C/C++ Example:

```
/* Get the timeout period. */

NIVXI_STATUS   ret;
INT32          actualtimo;
ret = WSgetTmo(&actualtimo);
```

BASIC Example:

```
' Get the timeout period.

Ret% = WSgetTmo% (actualtimo&)
```

WSrd / WSrdi / WSrdl

✔ VXI ☐ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS WSrd (UINT16 la, UINT8 *buf, UINT32 count, UINT16
modevalue, UINT32 *retcount);

NIVXI_STATUS WSrdi (INT16 la, UINT16 *buf, UINT32 count, UINT16
modevalue, UINT32 *retcount);

NIVXI_STATUS WSrdl (INT16 la, UINT32 *buf, UINT32 count, UINT16
modevalue, UINT32 *retcount);
```

BASIC Syntax

```
FUNCTION WSrd% (BYVAL la%, BYVAL buf$, BYVAL count&, BYVAL
modevalue%, retcount&)

FUNCTION WSrdi% (BYVAL la%, buf%(), BYVAL count&,
BYVAL modevalue%, retcount&)

FUNCTION WSrdl% (BYVAL la%, buf&(), BYVAL count&,
BYVAL modevalue%, retcount&)
```

Purpose

Transfers the specified number of bytes (or integers or longwords) from a message-based device into a specified local memory buffer, using the VXIbus Byte Transfer Protocol.

Parameters

Name	Direction	Description
la	Input	Logical address to read from the buffer
buf	Output	Read buffer
count	Input	Maximum number of bytes (or integers or longwords) to transfer
modevalue	Input	Transfer mode bit vector
retcount	Output	Number of bytes (or integers or longwords) actually transferred

Return Values

Return Values	Return Status Description
ret	Return status bit vector (See the bit description table in the *Description* section)

Description

The next table lists bits in the **modevalue** parameter.

Bit	Description
0	Not DOR 0 = Abort if not DOR 1 = Poll till DOR
1	END bit termination suppression 0 = Terminate transfer on END bit 1 = Do not terminate transfer on END
2	LF character termination 1 = Terminate transfer on LF bit 0 = Do not terminate transfer on LF
3	CR character termination 1 = Terminate transfer on CR bit 0 = Do not terminate transfer on CR
4	EOS character termination 1 = Terminate transfer on EOS bit 0 = Do not terminate transfer on EOS
8 to 15	EOS character (valid if EOS termination)

This table gives the meaning of each bit that is set to one (1) in the return value.

Bit	Name	Description
Bit 15 = 1 (Error Conditions)		
14	WRviol	Write Ready protocol violation during transfer
13	RRviol	Read Ready protocol violation during transfer
12	DORviol	Data Out Ready protocol violation
11	DIRviol	Data In Ready protocol violation
10	RdProtErr	Read protocol error
9	UnSupCom	Device did not recognize the command
8	TIMO	Timeout
7	BERR	Bus error occurred during transfer
6	MQE	Multiple query error occurred during transfer
5	InvalidLA	Invalid **la** specified
4	ForcedAbort	User abort occurred during I/O
Bit 15 = 0 (Successful Transfer)		
5	EOS	Termination character received
4	EOI	END bit received
3	DirDorAbort	Transfer aborted–device not DOR
2	TC	All bytes received
1	END	Any one of the termination received
0	IODONE	Command transfer successfully completed

C/C++ Example:

```
/* Read up to 30 bytes from a device at Logical Address 5. Poll until
   device is DOR. Terminate transfer on END bit only. */

NIVXI_STATUS   ret;
INT16          la;
UINT8          buf[30];
UINT32         count;
UINT16         modevalue;
UINT32         retcount;

la = 5;
count = 30L;
modevalue = 0x0001; /* Poll until DOR, terminate transfer on END. */
ret = WSrd (la, buf, count, modevalue, &retcount);
if (ret < 0)
         /* An error occurred during the buffer read. */;
```

BASIC Example:

```
' Read up to 30 bytes from a device at Logical Address 5. Poll
' until device is DOR. Terminate transfer on END bit only.

DIM buf AS STRING * 100
la% = 5
count& = 30&
modevalue% = &H0001 ' Poll until DOR, terminate transfer on END.
ret% = WSrd% (la%, buf$, count&, modevalue%, retcount&)
IF ret% < 0 THEN
         ' An error occurred during the buffer read.
END IF
```

WSrdf

✔ VXI ☐ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS WSrdf (UINT16 la, UINT8 *filename, UINT32 count,
UINT16 modevalue, UINT32 *retcount);
```

BASIC Syntax

```
FUNCTION WSrdf% (BYVAL la%, BYVAL filename$, BYVAL count&, BYVAL
modevalue%, retcount&)
```

Purpose

Reads the specified number of data bytes from a message-based device and writes them to the specified file using the VXIbus Byte Transfer Protocol and standard file I/O.

Parameters

Name	Direction	Description
la	Input	Logical address to read buffer from
filename	Input	Name of the file to read data into
count	Input	Maximum number of bytes to transfer
modevalue	Input	Transfer mode bit vector (See the *Description* section for bit descriptions)
retcount	Output	Number of bytes actually transferred

Return Values

Return Values	Return Status Description
ret	Return status bit vector (See the bit description table in the *Description* section)

Description

The next table describes the bits in the **modevalue** parameter

Bit	Description
0	Not DOR 0 = Abort if not DOR 1 = Poll till DOR
1	END bit termination suppression 0 = Terminate transfer on END bit 1 = Do not terminate transfer on END
2	LF character termination 1 = Terminate transfer on LF bit 0 = Do not terminate transfer on LF
3	CR character termination 1 = Terminate transfer on CR bit 0 = Do not terminate transfer on CR
4	EOS character termination 1 = Terminate transfer on EOS bit 0 = Do not terminate transfer on EOS
8 to 15	EOS character (valid if EOS termination)

The next table gives the meaning of each bit that is set to one (1) in the return value.

Bit	Name	Description
Bit 15 = 1 (Error Conditions)		
14	WRviol	Write Ready protocol violation during transfer
13	RRviol	Read Ready protocol violation during transfer
12	DORviol	Data Out Ready protocol violation
11	DIRviol	Data In Ready protocol violation
10	RdProtErr	Read protocol error
9	UnSupCom	Device did not recognize the command
8	TIMO	Timeout
7	BERR	Bus error occurred during transfer
6	MQE	Multiple query error occurred during transfer

Bit	Name	Description
5	InvalidLA	Invalid **la** specified
4	ForcedAbort	User abort occurred during I/O
1	FIOerr	Error reading or writing file
0	FOPENerr	Error opening file
Bit 15 = 0 (Successful Transfer)		
5	EOS	Termination character received
4	EOI	END bit received
3	DirDorAbort	Transfer aborted–Device not DOR
2	TC	All bytes received
1	END	Any one of the termination received
0	IODONE	Command transfer successfully completed

C/C++ Example:

```
/* Read 16 kilobytes (0x4000) from a device at Logical Address 5 into a
   file called "rdfile.dat." Poll until device is DOR. Terminate the
   transfer on END bit or line feed (LF). */

NIVXI_STATUS    ret;
INT8            *filename;
INT16           la;
UINT32          count;
UINT16          modevalue;
UINT32          retcount;

la = 5;
filename = "rdfile.dat";
count = 0x4000L;
modevalue = 0x0005;      /* Poll until DOR, terminate on END or LF. */
ret = WSrdf (la, filename, count, modevalue, &retcount);
if (ret < 0)
         /* An error occurred during the buffer read into the file. */
```

BASIC Example:

```
' Read 16 kilobytes (&H4000) from a device at Logical Address 5
' into a file called "rdfile.dat." Poll until device is DOR.
' Terminate the transfer on END bit or line feed (LF).

la% = 5
filename$ = "rdfile.dat"
count& = &H4000&
modevalue% = &H0005 ' Poll until DOR, terminate on END or LF.
ret% = WSrdf% (la%, filename$, count&, modevalue%, retcount&)
IF ret% < 0 THEN
         ' An error occurred during the buffer read into the file.
END IF
```

WSresp / WSLresp

✔ VXI ☐ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS WSresp (INT16 la, UINT16 *response);

NIVXI_STATUS WSLresp (INT16 la, UINT32 *response);
```

BASIC Syntax

```
FUNCTION WSresp% (BYVAL la%, response%)

FUNCTION WSLresp% (BYVAL la%, response&)
```

Purpose

Retrieves a response to a previously sent Word Serial Protocol query from a VXI message-based device.

☞ **Note:** ***This function is intended only for debugging purposes.***

Parameters

Name	Direction	Description
la	Input	Logical address of the message-based device
response	Output	16 or 32 bit location to store response

Return Values

Return Values	Return Status Description
ret	Return status bit vector (See the bit description table in the *Description* section)

Description

`WScmd` can send a query and automatically read a response. However, if it is necessary to break up the sending of the query and the reading of the response, you can use `WScmd` to send the query without reading the response and use `WSresp` to read the response.

The next table gives the meaning of each bit that is set to one (1) in the return value.

Bit	Name	Description
Bit 15 = 1 (Error Conditions)		
14	WRviol	Write Ready protocol violation during transfer
13	RRviol	Read Ready protocol violation during transfer
12	DORviol	Data Out Ready protocol violation
11	DIRviol	Data In Ready protocol violation
10	RdProtErr	Read protocol error
9	UnSupCom	Device did not recognize the command
7	BERR	Bus error occurred during transfer
6	MQE	Multiple query error occurred during transfer
5	InvalidLA	Invalid **la** specified
2	TIMO_RES	Timed out before response received
Bit 15 = 0 (Successful Transfer)		
0	IODONE	Command transfer successfully completed

C/C++ Example:

```
/* Send Read STB as a command and retrieve the response later. */

NIVXI_STATUS    ret;
INT16           la;
UINT16          cmd;
UINT16          respflag;
UINT16          response;

la = 5;
cmd = 0xcfff;
respflag = 0;      /* Do NOT read response. */
ret = WScmd (la, cmd, respflag, &response);
if ( ret < 0)
     /* An error occurred during WS command transfer. */;
else (
          ret = WSresp (la, &response);
          if (ret < 0)
             /* An error occurred during response retrieval. */;
}
```

BASIC Example:

```
' Send Read STB as a command and retrieve the response later.

la% = 5
cmd% = &HCFFF
respflag% = 0      ' Do NOT read response.
ret% = WScmd% (la%, cmd%, respflag%, response%)
IF ret% < 0 THEN
          ' Error occurred during WS command transfer.
ELSE
          ret% = WSresp% (la%, response%)
          IF ret% < 0 THEN
             ' Error occurred during response retrieval.
          END IF
END IF
```

WSSabort

☑ VXI ☐ VME ☑ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_STATUS WSSabort(UINT16 abortop);
```

Purpose

Aborts the Servant Word Serial operation(s) in progress.

Parameters

Name	Direction	Description
abortop	Input	The operation to abort bit vector (See the *Description* section for possible bit values.)

Return Values

Return Values	Return Status Description
0	Successfully aborted
-1	Servant Word Serial functions not supported
-2	Unable to abort

Description

The next table lists valid values for the **`abortop`** parameter.

Bit	Description
0	Abort `WSSwrt`
1	Abort `WSSrd`
2	Abort `WSSsendResp`
15	Initialize Word Serial Servant hardware. This includes aborting all Word Serial operations, clearing out errors, removing all pending Word Serial Servant interrupts, and disabling the interrupts.

C/C++ Example:

```
/* Abort WSSwrt. */

NIVXI_STATUS   ret;
UINT16         abortop;

abortop = (1<<0);
ret = WSSabort (abortop);
if (ret < 0)
         /* An error occurred during WSSabort. */;
```

WSSdisable

✔ VXI ☐ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_STATUS WSSdisable(void);
```

Purpose

Desensitizes the local CPU to interrupts generated when a Word Serial command is written to the Data Low register or when a response is read from the Data Low register.

Parameters

None

Return Values

Return Values	Return Status Description
0	Successful
-1	Servant Word Serial functions not supported

C/C++ Example:

```
/* Disable all the Servant Word Serial functions. */

NIVXI_STATUS   ret;

ret = WSSdisable();
if (ret < 0)
         /* An error occurred during WSSdisable. */;
```

WSSenable

✔ VXI ☐ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_STATUS WSSenable(void);
```

Purpose

Sensitizes the local CPU to interrupts generated when a Word Serial command is written to the Data Low register or when a response is read from the Data Low register.

Parameters

None

Return Values

Return Values	Return Status Description
0	Successful
-1	Servant Word Serial functions not supported

C/C++ Example:

```
/* Enable all the Servant Word Serial functions. */

NIVXI_STATUS   ret;

ret = WSSenable();
if (ret < 0)
         /* An error occurred during WSSenable. */;
```

WSsetTmo

✔ VXI ☐ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS WSsetTmo (INT32 timo, INT32 *actualtimo);
```

BASIC Syntax

```
FUNCTION WSsetTmo% (BYVAL timo&, actualtimo&)
```

Purpose

Sets the time period to wait before aborting a Word Serial, Longword Serial, or Extended Longword Serial Protocol transfer.

Parameters

Name	Direction	Description
timo	Input	Timeout period in milliseconds
actualtimo	Output	Actual timeout period set in milliseconds

Return Values

Return Values	Return Status Description
0	Successful

Description

`WSsetTmo` returns the actual timeout value set (the nearest timeout period possible greater than or equal to the timeout period specified).

C/C++ Example:

```
/* Set the timeout period to 2 seconds. */

NIVXI_STATUS   ret;
INT32          timo;
INT32          actualtimo;

timeout = 2000L;
ret = WSsetTmo (timo, &actualtimo);
```

BASIC Example:

```
' Set the timeout period to 2 seconds.

timo& = 2000&
ret% = WSsetTmo% (timo&, actualtimo&)
```

WSSnoResp / WSSLnoResp

✔ VXI ☐ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_STATUS WSSLnoResp(void);

NIVXI_STATUS WSSnoResp(void);
```

Purpose

Acknowledges a received Longword Serial Protocol command that requires no response and asserts the Write Ready (WR) bit in the local CPU Response register.

Parameters

None

Return Values

Return Values	Return Status Description
0	Successful
-1	Servant Word Serial functions not supported

Description

This function must be called after the processing of a Longword Serial Protocol command (queries are responded to with `WSSLsendResp`).

C/C++ Example:

```
/* Acknowledge the reception of a Longword Serial Protocol command that
   requires no response. */

NIVXI_STATUS   ret;

ret = WSSLnoResp ();
if (ret < 0)
         /* An error occurred during WSSLnoResp. */;
```

WSSrd / WSSrdi / WSSrdl

✔ VXI ☐ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_STATUS WSSrd(UINT8 *buf, UINT32 count, UINT16 mode);

NIVXI_STATUS WSSrdi(UINT16 *buf, UINT32 count, UINT16 mode);

NIVXI_STATUS WSSrdl(UINT32 *buf, UINT32 count, UINT16 mode);
```

Purpose

Posts a read operation to begin receiving the specified number of data bytes from a message-based Commander into a specified memory buffer using the VXIbus Byte Transfer Protocol.

Parameters

Name	Direction	Description
buf	Output	Read buffer
count	Input	Maximum number of bytes (or integers or longwords) to transfer
mode	Input	Transfer mode bit vector (See the *Description* section for possible bit values)

Return Values

Return Values	Return Status Description
0	Posted successfully
1	Posted successfully; will begin after a `WSSenable()`
-1	Servant Word Serial functions not supported
-2	`WSSrd` already in progress

Description

The next table lists bits that can be set using this function.

Bit	Description
0	DIR signal mode to Commander 0 = Do not send DIR signal to Commander 1 = Send DIR signal to Commander
15 to 1	Reserved (0)

C/C++ Example:

```
/* Read 10 bytes from the Commander. */

NIVXI_STATUS   ret;
UINT8          buf[100];
UINT32         count;
UINT16         mode;

count = 10L;
mode = 0x0000; /* Do not send DIR signal to Commander. */
ret = WSSrd (buf, count, mode);
if (ret < 0)
         /* An error occurred during WSSrd. */;
```

WSSsendResp / WSSLsendResp

✔ VXI ☐ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_STATUS WSSsendResp(UINT16 response);

NIVXI_STATUS WSSLsendResp(UINT32 response);
```

Purpose

Acknowledges a received Word Serial Protocol command that requires no response and asserts the WR bit in the local CPU Response register.

Parameters

None

Return Values

Return Values	Return Status Description
0	Successful
-1	Servant Word Serial functions not supported

Description

This function must be called after the processing of a Word Serial Protocol command. Queries are responded to with `WSSsendResp` and commands are acknowledged with `WSSLnoResp`.

C/C++ Example:

```
/* Acknowledge the reception of a Word Serial Protocol command that
   requires no response. */

NIVXI_STATUS   ret;

ret = WSSnoResp ();
if (ret < 0)
         /* An error occurred during WSSnoResp. */;
```

WSSwrt / WSSwrti / WSSwrtl

✔ VXI ☐ VME ✔ C/C++ ☐ BASIC

C/C++ Syntax

```
NIVXI_STATUS WSSwrt(UINT8 *buf, UINT32 count, UINT16 mode);

NIVXI_STATUS WSSwrti(UINT16 *buf, UINT32 count, UINT16 mode);

NIVXI_STATUS WSSwrtl(UINT32 *buf, UINT32 count, UINT16 mode);
```

Purpose

Posts the write operation to transfer the specified number of bytes (or integers or longwords) from a specified memory buffer to the message-based Commander using the VXIbus Byte Transfer Protocol.

Parameters

Name	Direction	Description
buf	Input	Write buffer
count	Input	Maximum number of bytes (or integers or longwords) to transfer
mode	Input	Mode of transfer (bit vector) (See the *Description* section for possible bit values)

Return Values

Return Values	Return Status Description
0	Posted successfully
1	Posted successfully; will begin after a `WSSenable()`
-1	Servant Word Serial functions not supported
-2	`WSSwrt` already in progress

Description

The next table lists bits that can be set using this function.

Bit	Description
0	DOR signal mode to Commander (if enabled) 0 = Do not send DOR signal to Commander 1 = Send DOR signal to Commander
1	END bit termination with last byte 0 = Do not send END with the last byte 1 = Send END with the last byte
2 to 15	Reserved

C/C++ Example:

```
/* Write 6 bytes to the Commander. */

NIVXI_STATUS   ret;
UINT8          *buf;
UINT32         count;
UINT16         mode;

buf = "1.0422";
count = 6L;
mode = 0x0002;     /* Send END with the last byte. */
ret = WSSwrt (buf, count, mode);
if (ret < 0)
         /* An error occurred during WSSwrt. */;
```

WStrg

✔ VXI ☐ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS WStrg (INT16 la);
```

BASIC Syntax

```
FUNCTION WStrg% (BYVAL la%)
```

Purpose

Sends the Word Serial *Trigger* command to a message-based device.

Parameters

Name	Direction	Description
la	Input	Logical address of the message-based device

Return Values

Return Values	Return Status Description
ret	Return status bit vector (See the bit description table in the *Description* section.)

Description

The next table gives the meaning of each bit that is set to one (1) in the return value.

Bit	Name	Description
Bit 15 = 1 (Error Conditions)		
14	WRviol	Write Ready protocol violation during transfer
13	RRviol	Read Ready protocol violation during transfer
12	DORviol	Data Out Ready protocol violation
11	DIRviol	Data In Ready protocol violation

Bit	Name	Description
10	RdProtErr	Read protocol error
9	UnSupCom	Device did not recognize the command
7	BERR	Bus error occurred during transfer
6	MQE	Multiple query error occurred during transfer
5	InvalidLA	Invalid **la** specified
4	ForcedAbort	User abort occurred during I/O
1	TIMO_SEND	Timed out before able to send *Clear*
Bit 15 = 0 (Successful Transfer)		
0	IODONE	Command transfer successfully completed

C/C++ Example:

```
/* Send Trigger command to Logical Address 5. */

NIVXI_STATUS   ret;
INT16          la;

la = 5;
ret = WStrg (la);
if (ret < 0)
         /* An error occurred during the command transfer. */;
```

BASIC Example:

```
' Send Trigger command to Logical Address 5.

la% = 5
ret% = WStrg% (la%)
IF ret% < 0 THEN
          ' An error occurred during the command transfer.
END IF
```

WSwrt / WSwrti / WSwrtl

✔ VXI ☐ VME ✔ C/C++ ✔ BASIC

C/C++ Syntax

```
NIVXI_STATUS WSwrt (INT16 la, UINT8 *buf, UINT32 count, UINT16
modevalue, UINT32 *retcount);

NIVXI_STATUS WSwrti (INT16 la, UINT16 *buf, UINT32 count, UINT16
modevalue, UINT32 * retcount);

NIVXI_STATUS WSwrtl (INT16 la, UINT32 *buf, UINT32 count, UINT16
modevalue, UINT32 * retcount);
```

BASIC Syntax

```
FUNCTION WSwrt% (BYVAL la%, BYVAL buf$, BYVAL count&, BYVAL
modevalue%, retcount&)

FUNCTION WSwrti% (BYVAL la%, BYVAL buf%(), BYVAL count&, BYVAL
modevalue%, retcount&)

FUNCTION WSwrtl% (BYVAL la%, BYVAL buf&(), BYVAL count&, BYVAL
modevalue%, retcount&)
```

Purpose

Transfers the specified number of bytes (or integers or longwords) from a specified local memory buffer to a message-based device, using the VXIbus Byte Transfer Protocol.

Parameters

Name	Direction	Description
la	Input	VXI logical address to which to write the buffer
buf	Input	Write buffer
count	Input	Maximum number of bytes (or integers or longwords) to transfer
modevalue	Input	Mode of transfer (bit vector) (See the *Description* section for more information)
retcount	Output	Number of bytes (or integers or longwords) actually transferred

Return Values

Return Values	Return Status Description
ret	Return status bit vector (See the bit description table in the *Description* section)

Description

The next table lists fields and bits that can be set using this function.

Bit	Description
0	0 = Abort if device is not DIR 1 = Poll until device is DIR
1	1 = Set END bit on the last byte of transfer 0 = Clear END bit on the last byte of transfer

The next table gives the meaning of each bit that is set to one (1) in the return value.

Bit	Name	Description
Bit 15 = 1 (Error Conditions)		
14	WRviol	Write Ready protocol violation during transfer
13	RRviol	Read Ready protocol violation during transfer
12	DORviol	Data Out Ready protocol violation
11	DIRviol	Data In Ready protocol violation
10	RdProtErr	Read protocol error
9	UnSupCom	Device did not recognize the command
8	TIMO	Timeout
7	BERR	Bus error occurred during transfer
6	MQE	Multiple query error occurred during transfer
5	InvalidLA	Invalid **la** specified
4	ForcedAbort	User abort occurred during I/O

Bit	Name	Description
Bit 15 = 0 (Successful Transfer)		
3	DirDorAbort	Transfer aborted–Device not DOR
2	TC	All bytes received
1	END	Any one of the termination received
0	IODONE	Command transfer successfully completed

C/C++ Example:

```
/* Write the 14-byte ASCII command "VXI:CONF:NUMB?" to a device at
   Logical Address 5. Poll until device is DIR, and send END with the
   last byte. */

NIVXI_STATUS   ret;
INT16          la;
UINT8          *buf;
UINT32         count;
UINT16         modevalue;
UINT32         retcount;

la = 5;
buf = "VXI:CONF:NUMB?";
count = strlen(buf);
modevalue = 0x0003;      /* Poll until DIR; send END with last byte. */
ret = WSwrt (la, buf, count, modevalue, &retcount);
if (ret < 0)
         /* An error occurred during the buffer write. */;
```

BASIC Example:

```
' Write the 14-byte ASCII command "VXI:CONF:NUMB?" to a device at
' Logical Address 5. Poll until device is DIR, and send END with
' the last byte.

la% = 5
buf$ = "VXI:CONF:NUMB?"
count& = StringLength% (buf$)
modevalue% = &H0003      ' Poll until DIR; send END with last byte.
ret% = WSwrt% (la%, buf$, count&, modevalue%, retcount&)
IF ret% < 0 THEN
         ' An error occurred during the buffer write.
END IF
```

WSwrtf

☑ VXI ☐ VME ☑ C/C++ ☑ BASIC

C/C++ Syntax

```
NIVXI_STATUS WSwrtf (INT16 la, INT8 filename, UINT32 count, UINT16
modevalue, UINT32 *retcount);
```

BASIC Syntax

```
FUNCTION WSwrtf% (BYVAL la%, BYVAL filename$, BYVAL count&,
BYVAL modevalue%, retcount&)
```

Purpose

Transfers up to the specified number of data bytes from the specified file to a message-based device using the VXIbus Byte Transfer Protocol and standard file I/O.

Parameters

Name	Direction	Description
la	Input	VXI logical address to write buffer to
filename	Input	Name of the file to write data from
count	Input	Maximum number of bytes to transfer
modevalue	Input	Mode of transfer (bit vector)
retcount	Output	Number of bytes actually transferred

Return Values

Return Values	Return Status Description
ret	Return status bit vector (See the bit description table in the *Description* section)

Description

Bit	Description
0	0 = Abort if device is not DIR 1 = Poll until device is DIR
1	1 = Set END bit on the last byte of transfer 0 = Clear END bit on the last byte of transfer

The next table gives the meaning of each bit that is set to one (1) in the return value.

Bit	Name	Description
Bit 15 = 1 (Error Conditions)		
14	WRviol	Write Ready protocol violation during transfer
13	RRviol	Read Ready protocol violation during transfer
12	DORviol	Data Out Ready protocol violation
11	DIRviol	Data In Ready protocol violation
10	RdProtErr	Read protocol error
9	UnSupCom	Device did not recognize the command
8	TIMO	Timeout
7	BERR	Bus error occurred during transfer
6	MQE	Multiple query error occurred during transfer
5	InvalidLA	Invalid **la** specified
4	ForcedAbort	User abort occurred during I/O
1	FIOerr	Error reading or writing file
0	FOPENerr	Error opening file
Bit 15 = 0 (Successful Transfer)		
3	DirDorAbort	Transfer aborted–device not DOR
2	TC	All bytes sent
1	END	The END bit was sent
0	IODONE	Command transfer successfully completed

C/C++ Example:

```
/* Write 16 kilobytes (0x4000) to a device at Logical Address 5 from the
   file, "wrtfile.dat." Poll until device is DIR, and send END with the
   last byte. */

NIVXI_STATUS   ret;
INT8           *filename;
INT16          la;
UINT32         count;
UINT16         modevalue;
UINT32         retcount;

la = 5;
filename = "wrtfile.dat";
count = 0x4000L;
modevalue = 0x0003;      /* Send END, wait until DIR if not
 already DIR. */
ret = WSwrtf (la, filename, count, modevalue, &retcount);
if (ret < 0)
         /* An error occurred during the buffer write. */;
```

BASIC Example:

```
' Write 16 kilobytes (&H4000&) to a device at Logical Address 5
' from the file "wrtfile.dat." Poll until device is DIR, and send
' END with the last byte.

la% = 5
filename$ = "wrtfile.dat"
count& = &H4000&
modevalue% = &H0003 ' Send END, wait until DIR if not already DIR.
ret% = WSwrtf% (la%, filename$, count&, modevalue%, retcount&)
IF ret% < 0 THEN
         ' An error occurred during the buffer write.
END IF
```

Function Classification Reference

Appendix

A

This appendix contains two tables you can use as a quick reference. Table A-1, *Function Listing by Group*, lists the NI-VXI functions by their group association. This arrangement can help you determine easily which functions are available within each group. Table A-2, *Function Listing by Name*, lists each function alphabetically. You can refer to this table if you don't remember the group association of a particular function. Both tables use checkmarks to represent whether a VXI function also applies to VME and also whether it is associated with C/C++ and/or BASIC.

Table A-1. Function Listing by Group

Group	Function	VXI	VME	C/C++	BASIC
System Configuration	`CloseVXIlibrary`	✔	✔	✔	✔
	`CreateDevInfo`	✔	✔	✔	✔
	`FindDevLA`	✔	✔	✔	✔
	`GetDevInfo`	✔	✔	✔	
	`GetDevInfoLong`	✔	✔	✔	✔
	`GetDevInfoShort`	✔	✔	✔	✔
	`GetDevInfoStr`	✔	✔	✔	✔
	`InitVXIlibrary`	✔	✔	✔	✔
	`SetDevInfo`	✔	✔	✔	
	`SetDevInfoLong`	✔	✔	✔	✔
	`SetDevInfoShort`	✔	✔	✔	✔
	`SetDevInfoStr`	✔	✔	✔	✔

Table A-1. Function Listing by Group

Group	Function	VXI	VME	C/C++	BASIC
Commander Word Serial Protocol	WSabort	✔		✔	✔
	WSclr	✔		✔	✔
	WScmd / WSEcmd/ WSLcmd	✔		✔	✔
	WSgetTmo	✔		✔	✔
	WSrd / WSrdi / WSrdl	✔		✔	✔
	WSrdf	✔		✔	✔
	WSresp/WSLresp	✔		✔	✔
	WSsetTmo	✔		✔	✔
	WStrg	✔		✔	✔
	WSwrt / WSwrti / WSwrtl	✔		✔	✔
	WSwrtf	✔		✔	✔
Servant Word Serial Protocol	DefaultWSScmdHandler	✔		✔	
	DefaultWSSEcmdHandler	✔		✔	
	DefaultWSSLcmdHandler	✔		✔	
	DefaultWSSrdHandler	✔		✔	
	DefaultWSSwrtHandler	✔		✔	
	GenProtError	✔		✔	
	GetWSScmdHandler	✔		✔	
	GetWSSEcmdHandler	✔		✔	
	GetWSSLcmdHandler	✔		✔	
	GetWSSrdHandler	✔		✔	
	GetWSSwrtHandler	✔		✔	
	RespProtError	✔		✔	
	SetWSScmdHandler	✔		✔	
	SetWSSEcmdHandler	✔		✔	
	SetWSSLcmdHandler	✔		✔	
	SetWSSrdHandler	✔		✔	

Table A-1. Function Listing by Group

Group	Function	VXI	VME	C/C++	BASIC
Servant Word Serial Protocol (continued)	SetWSSwrtHandler	✔		✔	
	WSSabort	✔		✔	
	WSSdisable	✔		✔	
	WSSenable	✔		✔	
	WSSnoResp /WSSLnoResp	✔		✔	
	WSSrd / WSSrdi / WSSrdl	✔		✔	
	WSSsendResp / WSSLsendResp	✔		✔	
	WSSwrt / WSSwrti / WSSwrtl	✔		✔	
High-Level VXI/VMEbus Access	VXIin	✔	✔	✔	✔
	VXIinReg	✔		✔	✔
	VXImove	✔	✔	✔	✔
	VXIout	✔	✔	✔	✔
	VXIoutReg	✔		✔	✔
Low-Level VXI/VMEbus Access	GetByteOrder	✔	✔	✔	✔
	GetContext	✔	✔	✔	✔
	GetPrivilege	✔	✔	✔	✔
	GetVXIbusStatus	✔	✔	✔	
	GetVXIbusStatusInd	✔	✔	✔	✔
	GetWindowRange	✔	✔	✔	✔
	MapVXIAddress	✔	✔	✔	✔
	MapVXIAddressSize	✔	✔	✔	✔
	SetByteOrder	✔	✔	✔	✔
	SetContext	✔	✔	✔	✔
	SetPrivilege	✔	✔	✔	✔
	UnMapVXIAddress	✔	✔	✔	✔
	VXIpeek	✔	✔	✔	✔
	VXIpoke	✔	✔	✔	✔

Table A-1. Function Listing by Group

Group	Function	VXI	VME	C/C++	BASIC
Local Resource Access	GetMyLa	✔	✔	✔	✔
	ReadMODID	✔		✔	✔
	SetMODID	✔		✔	✔
	VXIinLR	✔	✔	✔	✔
	VXImemAlloc	✔	✔	✔	✔
	VXImemCopy	✔	✔	✔	✔
	VXImemFree	✔	✔	✔	✔
	VXIoutLR	✔	✔	✔	✔
VXI Signal	DefaultSignalHandler	✔	✔	✔	✔
	DisableSignalInt	✔		✔	✔
	EnableSignalInt	✔		✔	✔
	GetSignalHandler	✔	✔	✔	
	RouteSignal	✔	✔	✔	✔
	SetSignalHandler	✔	✔	✔	
	SignalDeq	✔	✔	✔	✔
	SignalEnq	✔	✔	✔	✔
	SignalJam	✔	✔	✔	✔
	WaitForSignal	✔	✔	✔	✔
VXI/VME Interrupt	AcknowledgeVXIint	✔	✔	✔	✔
	AssertVXIint	✔	✔	✔	✔
	DeAssertVXIint	✔	✔	✔	✔
	DefaultVXIintHandler	✔	✔	✔	✔
	DisableVXIint	✔	✔	✔	✔
	DisableVXItoSignalInt	✔	✔	✔	✔
	EnableVXIint	✔	✔	✔	✔
	EnableVXItoSignalInt	✔	✔	✔	✔

Table A-1. Function Listing by Group

Group	Function	VXI	VME	C/C++	BASIC
VXI/VME Interrupt (continued)	GetVXIintHandler	✔	✔	✔	
	RouteVXIint	✔	✔	✔	✔
	SetVXIintHandler	✔	✔	✔	
	VXIintAcknowledgeMode	✔	✔	✔	✔
Triggers	AcknowledgeTrig	✔		✔	✔
	DefaultTrigHandler	✔		✔	✔
	DefaultTrigHandler2	✔		✔	✔
	DisableTrigSense	✔		✔	✔
	EnableTrigSense	✔		✔	✔
	GetTrigHandler	✔		✔	
	MapTrigToTrig	✔		✔	✔
	SetTrigHandler	✔		✔	
	SrcTrig	✔		✔	✔
	TrigAssertConfig	✔		✔	✔
	TrigCntrConfig	✔		✔	✔
	TrigExtConfig	✔		✔	✔
	TrigTickConfig	✔		✔	✔
	UnMapTrigToTrig	✔		✔	✔
	WaitForTrig	✔		✔	✔
System Interrupt Handler	AssertSysreset	✔	✔	✔	✔
	DefaultACfailHandler	✔	✔	✔	✔
	DefaultBusErrorHandler	✔	✔	✔	✔
	DefaultSoftResetHandler	✔		✔	✔
	DefaultSysfailHandler	✔	✔	✔	✔
	DefaultSysresetHandler	✔	✔	✔	✔
	DisableACfail	✔	✔	✔	✔

Table A-1. Function Listing by Group

Group	Function	VXI	VME	C/C++	BASIC
System Interrupt Handler (continued)	DisableSoftReset	✔		✔	✔
	DisableSysfail	✔	✔	✔	✔
	DisableSysreset	✔	✔	✔	✔
	EnableACfail	✔	✔	✔	✔
	EnableSoftReset	✔		✔	✔
	EnableSysfail	✔	✔	✔	✔
	EnableSysreset	✔	✔	✔	✔
	GetACfailHandler	✔	✔	✔	
	GetBusErrorHandler	✔	✔	✔	
	GetSoftResetHandler	✔		✔	
	GetSysfailHandler	✔	✔	✔	
	GetSysresetHandler	✔	✔	✔	
	SetACfailHandler	✔	✔	✔	
	SetBusErrorHandler	✔	✔	✔	
	SetSoftResetHandler	✔		✔	
	SetSysfailHandler	✔	✔	✔	
	SetSysresetHandler	✔	✔	✔	
VXI/VMEbus Extender	MapECLtrig	✔		✔	✔
	MapTTLtrig	✔		✔	✔
	MapUtilBus	✔	✔	✔	✔
	MapVXIint	✔	✔	✔	✔

Table A-2. Function Listing by Name

Function	Group	VXI	VME	C/C++	BASIC
AcknowledgeTrig	Triggers	✔		✔	✔
AcknowledgeVXIint	VXI/VME Interrupt	✔	✔	✔	✔
AssertSysreset	System Interrupt Handler	✔	✔	✔	✔
AssertVXIint	VXI/VME Interrupt	✔	✔	✔	✔
CloseVXIlibrary	System Configuration	✔	✔	✔	✔
CreateDevInfo	System Configuration	✔	✔	✔	✔
DeAssertVXIint	VXI/VME Interrupt	✔	✔	✔	✔
DefaultACfailHandler	System Interrupt Handler	✔	✔	✔	✔
DefaultBusErrorHandler	System Interrupt Handler	✔	✔	✔	✔
DefaultSignalHandler	VXI Signal	✔	✔	✔	✔
DefaultSoftResetHandler	System Interrupt Handler	✔		✔	✔
DefaultSysfailHandler	System Interrupt Handler	✔	✔	✔	✔
DefaultSysresetHandler	System Interrupt Handler	✔	✔	✔	✔
DefaultTrigHandler	Triggers	✔		✔	✔
DefaultTrigHandler2	Triggers	✔		✔	✔
DefaultVXIintHandler	VXI/VME Interrupt	✔	✔	✔	✔
DefaultWSScmdHandler	Servant Word Serial Protocol	✔		✔	
DefaultWSSEcmdHandler	Servant Word Serial Protocol	✔		✔	
DefaultWSSLcmdHandler	Servant Word Serial Protocol	✔		✔	
DefaultWSSrdHandler	Servant Word Serial Protocol	✔		✔	
DefaultWSSwrtHandler	Servant Word Serial Protocol	✔		✔	

Table A-2. Function Listing by Name

Function	Group	VXI	VME	C/C++	BASIC
DisableACfail	System Interrupt Handler	✔	✔	✔	✔
DisableSignalInt	VXI Signal	✔		✔	✔
DisableSoftReset	System Interrupt Handler	✔		✔	✔
DisableSysfail	System Interrupt Handler	✔	✔	✔	✔
DisableSysreset	System Interrupt Handler	✔	✔	✔	✔
DisableTrigSense	Triggers	✔		✔	✔
DisableVXIint	VXI/VME Interrupt	✔	✔	✔	✔
DisableVXItoSignalInt	VXI/VME Interrupt	✔	✔	✔	✔
EnableACfail	System Interrupt Handler	✔	✔	✔	✔
EnableSignalInt	VXI Signal	✔		✔	✔
EnableSoftReset	System Interrupt Handler	✔		✔	✔
EnableSysfail	System Interrupt Handler	✔	✔	✔	✔
EnableSysreset	System Interrupt Handler	✔	✔	✔	✔
EnableTrigSense	Triggers	✔		✔	✔
EnableVXIint	VXI/VME Interrupt	✔	✔	✔	✔
EnableVXItoSignalInt	VXI/VME Interrupt	✔	✔	✔	✔
FindDevLA	System Configuration	✔	✔	✔	✔
GenProtError	Servant Word Serial Protocol	✔		✔	
GetACfailHandler	System Interrupt Handler	✔	✔	✔	
GetBusErrorHandler	System Interrupt Handler	✔	✔	✔	
GetByteOrder	Low-Level VXI/VMEbus Access	✔	✔	✔	✔

Table A-2. Function Listing by Name

Function	Group	VXI	VME	C/C++	BASIC
GetContext	Low-Level VXI/VMEbus Access	✔	✔	✔	✔
GetDevInfo	System Configuration	✔	✔	✔	
GetDevInfoLong	System Configuration	✔	✔	✔	✔
GetDevInfoShort	System Configuration	✔	✔	✔	✔
GetDevInfoStr	System Configuration	✔	✔	✔	✔
GetMyLa	Local Resource Access	✔	✔	✔	✔
GetPrivilege	Low-Level VXI/VMEbus Access	✔	✔	✔	✔
GetSignalHandler	VXI Signal	✔	✔	✔	
GetSoftResetHandler	System Interrupt Handler	✔		✔	
GetSysfailHandler	System Interrupt Handler	✔	✔	✔	
GetSysresetHandler	System Interrupt Handler	✔	✔	✔	
GetTrigHandler	Triggers	✔		✔	
GetVXIbusStatus	Low-Level VXI/VMEbus Access	✔	✔	✔	
GetVXIbusStatusInd	Low-Level VXI/VMEbus Access	✔	✔	✔	✔
GetVXIintHandler	VXI/VME Interrupt	✔	✔	✔	
GetWindowRange	Low-Level VXI/VMEbus Access	✔	✔	✔	✔
GetWSScmdHandler	Servant Word Serial Protocol	✔		✔	
GetWSSEcmdHandler	Servant Word Serial Protocol	✔		✔	
GetWSSLcmdHandler	Servant Word Serial Protocol	✔		✔	
GetWSSrdHandler	Servant Word Serial Protocol	✔		✔	
GetWSSwrtHandler	Servant Word Serial Protocol	✔		✔	

Table A-2. Function Listing by Name

Function	Group	VXI	VME	C/C++	BASIC
InitVXIlibrary	System Configuration	✔	✔	✔	✔
MapECLtrig	VXI/VMEbus Extender	✔		✔	✔
MapTrigToTrig	Triggers	✔		✔	✔
MapTTLtrig	VXI/VMEbus Extender	✔		✔	✔
MapUtilBus	VXI/VMEbus Extender	✔	✔	✔	✔
MapVXIAddress	Low-Level VXI/VMEbus Access	✔	✔	✔	✔
MapVXIAddressSize	Low-Level VXI/VMEbus Access	✔	✔	✔	✔
MapVXIint	VXI/VMEbus Extender	✔	✔	✔	✔
ReadMODID	Local Resource Access	✔		✔	✔
RespProtError	Servant Word Serial Protocol	✔		✔	
RouteSignal	VXI Signal	✔	✔	✔	✔
RouteVXIint	VXI/VME Interrupt	✔	✔	✔	✔
SetACfailHandler	System Interrupt Handler	✔	✔	✔	
SetBusErrorHandler	System Interrupt Handler	✔	✔	✔	
SetByteOrder	Low-Level VXI/VMEbus Access	✔	✔	✔	✔
SetContext	Low-Level VXI/VMEbus Access	✔	✔	✔	✔
SetDevInfo	System Configuration	✔	✔	✔	
SetDevInfoLong	System Configuration	✔	✔	✔	✔
SetDevInfoShort	System Configuration	✔	✔	✔	✔
SetDevInfoStr	System Configuration	✔	✔	✔	✔
SetMODID	Local Resource Access	✔		✔	✔

Table A-2. Function Listing by Name

Function	Group	VXI	VME	C/C++	BASIC
SetPrivilege	Low-Level VXI/VMEbus Access	✔	✔	✔	✔
SetSignalHandler	VXI Signal	✔	✔	✔	
SetSoftResetHandler	System Interrupt Handler	✔		✔	
SetSysfailHandler	System Interrupt Handler	✔	✔	✔	
SetSysresetHandler	System Interrupt Handler	✔	✔	✔	
SetTrigHandler	Triggers	✔		✔	
SetVXIintHandler	VXI/VME Interrupt	✔	✔	✔	
SetWSScmdHandler	Servant Word Serial Protocol	✔		✔	
SetWSSEcmdHandler	Servant Word Serial Protocol	✔		✔	
SetWSSLcmdHandler	Servant Word Serial Protocol	✔		✔	
SetWSSrdHandler	Servant Word Serial Protocol	✔		✔	
SetWSSwrtHandler	Servant Word Serial Protocol	✔		✔	
SignalDeq	VXI Signal	✔	✔	✔	✔
SignalEnq	VXI Signal	✔	✔	✔	✔
SignalJam	VXI Signal	✔	✔	✔	✔
SrcTrig	Triggers	✔		✔	✔
TrigAssertConfig	Triggers	✔		✔	✔
TrigCntrConfig	Triggers	✔		✔	✔
TrigExtConfig	Triggers	✔		✔	✔
TrigTickConfig	Triggers	✔		✔	✔
UnMapTrigToTrig	Triggers	✔		✔	✔
UnMapVXIAddress	Low-Level VXI/VMEbus Access	✔	✔	✔	✔

Table A-2. Function Listing by Name

Function	Group	VXI	VME	C/C++	BASIC
VXIin	High-Level VXI/VMEbus Access	✔	✔	✔	✔
VXIinLR	Local Resource Access	✔	✔	✔	✔
VXIinReg	High-Level VXI/VMEbus Access	✔		✔	✔
VXIintAcknowledgeMode	VXI/VME Interrupt	✔	✔	✔	✔
VXImemAlloc	Local Resource Access	✔	✔	✔	✔
VXImemCopy	Local Resource Access	✔	✔	✔	✔
VXImemFree	Local Resource Access	✔	✔	✔	✔
VXImove	High-Level VXI/VMEbus Access	✔	✔	✔	✔
VXIout	High-Level VXI/VMEbus Access	✔	✔	✔	✔
VXIoutLR	Local Resource Access	✔	✔	✔	✔
VXIoutReg	High-Level VXI/VMEbus Access	✔		✔	✔
VXIpeek	Low-Level VXI/VMEbus Access	✔	✔	✔	✔
VXIpoke	Low-Level VXI/VMEbus Access	✔	✔	✔	✔
WaitForSignal	VXI Signal	✔	✔	✔	✔
WaitForTrig	Triggers	✔		✔	✔
WSabort	Commander Word Serial Protocol	✔		✔	✔
WSclr	Commander Word Serial Protocol	✔		✔	✔
WScmd/ WSEcmd/ WSLcmd	Commander Word Serial Protocol	✔		✔	✔
WSgetTmo	Commander Word Serial Protocol	✔		✔	✔
WSresp/ WSLresp	Commander Word Serial Protocol	✔		✔	✔

Table A-2. Function Listing by Name

Function	Group	VXI	VME	C/C++	BASIC
WSrd / WSrdi / WSrdl	Commander Word Serial Protocol	✔		✔	✔
WSrdf	Commander Word Serial Protocol	✔		✔	✔
WSSabort	Servant Word Serial Protocol	✔		✔	
WSSdisable	Servant Word Serial Protocol	✔		✔	
WSSenable	Servant Word Serial Protocol	✔		✔	
WSsetTmo	Commander Word Serial Protocol	✔		✔	✔
WSSnoResp/ WSSLnoResp	Servant Word Serial Protocol	✔		✔	
WSSrd / WSSrdi / WSSrdl	Servant Word Serial Protocol	✔		✔	
WSSsendResp/ WSSLsendResp	Servant Word Serial Protocol	✔		✔	
WSSwrt / WSSwrti / WSSwrtl	Servant Word Serial Protocol	✔		✔	
WStrg	Commander Word Serial Protocol	✔		✔	✔
WSwrt / WSwrti / WSwrtl	Commander Word Serial Protocol	✔		✔	✔
WSwrtf	Commander Word Serial Protocol	✔		✔	✔

Appendix

B

Customer Communication

For your convenience, this appendix contains forms to help you gather the information necessary to help us solve technical problems you might have as well as a form you can use to comment on the product documentation. Filling out a copy of the *Technical Support Form* before contacting National Instruments helps us help you better and faster.

National Instruments provides comprehensive technical assistance around the world. In the U.S. and Canada, applications engineers are available Monday through Friday from 8:00 a.m. to 6:00 p.m. (central time). In other countries, contact the nearest branch office. You may fax questions to us at any time.

Electronic Services

Bulletin Board Support

National Instruments has BBS and FTP sites dedicated for 24-hour support with a collection of files and documents to answer most common customer questions. From these sites, you can also download the latest instrument drivers, updates, and example programs. For recorded instructions on how to use the bulletin board and FTP services and for BBS automated information, call (512) 795-6990. You can access these services at:

United States: (512) 794-5422 or (800) 327-3077
Up to 14,400 baud, 8 data bits, 1 stop bit, no parity

United Kingdom: 01635 551422
Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

France: 1 48 65 15 59
Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

FTP Support

To access our FTP site, log on to our Internet host, `ftp.natinst.com`, as anonymous and use your Internet address, such as `joesmith@anywhere.com`, as your password. The support files and documents are located in the `/support` directories.

FaxBack Support

FaxBack is a 24-hour information retrieval system containing a library of documents on a wide range of technical information. You can access FaxBack from a touch-tone telephone at (512) 418-1111.

E-Mail Support (currently U.S. only)

You can submit technical support questions to the appropriate applications engineering team through e-mail at the Internet addresses listed below. Remember to include your name, address, and phone number so we can contact you with solutions and suggestions.

GPIB: `gpib.support@natinst.com`
DAQ: `daq.support@natinst.com`
VXI: `vxi.support@natinst.com`
LabWindows: `lw.support@natinst.com`
LabVIEW: `lv.support@natinst.com`
HiQ: `hiq.support@natinst.com`
VISA: `visa.support@natinst.com`
Lookout: `lookout.support@natinst.com`

Telephone and Fax Support

National Instruments has branch offices all over the world. Use the list below to find the technical support number for your country. If there is no National Instruments office in your country, contact the source from which you purchased your software to obtain support.

	 Telephone	 Fax
Australia	03 9 879 9422	03 9 879 9179
Austria	0662 45 79 90 0	0662 45 79 90 19
Belgium	02 757 00 20	02 757 03 11
Canada (Ontario)	519 622 9310	
Canada (Quebec)	514 694 8521	514 694 4399
Denmark	45 76 26 00	45 76 26 02
Finland	90 527 2321	90 502 2930
France	1 48 14 24 24	1 48 14 24 14
Germany	089 741 31 30	089 714 60 35
Hong Kong	2645 3186	2686 8505
Italy	02 413091	02 41309215
Japan	03 5472 2970	03 5472 2977
Korea	02 596 7456	02 596 7455
Mexico	95 800 010 0793	5 520 3282
Netherlands	0348 433466	0348 430673
Norway	32 84 84 00	32 84 86 00
Singapore	2265886	2265887
Spain	91 640 0085	91 640 0533
Sweden	08 730 49 70	08 730 43 70
Switzerland	056 200 51 51	056 200 51 55
Taiwan	02 377 1200	02 737 4644
U.K.	01635 523545	01635 523154

Documentation Comment Form

National Instruments encourages you to comment on the documentation supplied with our products. This information helps us provide quality products to meet your needs.

Title: *NI-VXI™ Programmer Reference Manual*

Edition Date: July 1996

Part Number: 321272A-01

Please comment on the completeness, clarity, and organization of the manual.

_____

_____

_____

_____

_____

_____

_____

If you find errors in the manual, please record the page numbers and describe the errors.

_____

_____

_____

_____

_____

_____

_____

Thank you for your help.

Name _____

Title _____

Company _____

Address _____

_____

Phone (____) _____

Mail to: Technical Publications
National Instruments Corporation
6504 Bridge Point Parkway
Austin, TX 78730-5039

Fax to: Technical Publications
National Instruments Corporation
(512) 794-5678

Glossary

Prefix	Meaning	Value
n-	nano-	10^{-9}
m-	milli-	10^{-3}
K-	kilo-	10^{3}
M-	mega-	10^{6}
G-	giga-	10^{9}

A

A16 space — One of the VXIbus address spaces. Equivalent to the VME 64 KB short address space. In VXI, the upper 16 KB of A16 space is allocated for use by VXI devices configuration registers. This 16 KB region is referred to as VXI configuration space.

A24 space — One of the VXIbus address spaces. Equivalent to the VME 16 MB standard address space.

A32 space — One of the VXIbus address spaces. Equivalent to the VME 4 GB extended address space.

ACFAIL* — A VMEbus backplane signal that is asserted when a power failure has occurred (either AC line source or power supply malfunction), or if it is necessary to disable the power supply (such as for a high temperature condition).

address — A numeric value that identifies a specific location (or series of locations) in memory.

address modifier	One of six signals in the VMEbus specification used by VMEbus masters to indicate the address space and mode (supervisory/nonprivileged, data/program/block) in which a data transfer is to take place.
address space	A set of 2^n memory locations differentiated from other such sets in VXI/VMEbus systems by six signal lines known as address modifiers. *n* is the number of address lines required to uniquely specify a byte location in a given space. Valid numbers for *n* are 16, 24, and 32.
address window	A range of address space that can be accessed from the application program.
ANSI	American National Standards Institute
ASCII	American Standard Code for Information Interchange. A 7-bit standard code adopted to facilitate the interchange of data among various types of data processing and data communications equipment.
ASIC	Application-Specific Integrated Circuit (a custom chip)
asserted	A signal in its active true state.
asynchronous	Not synchronized; not controlled by periodic time signals, and therefore unpredictable with regard to the timing of execution of commands.
ASYNC Protocol	A two-device, two-line handshake trigger protocol using two consecutive even/odd trigger lines (a source/acceptor line and an acknowledge line).

B

backplane	An assembly, typically a PCB, with 96-pin connectors and signal paths that bus the connector pins. A C-size VXIbus system will have two sets of bused connectors called the J1 and J2 backplanes. A D-size VXIbus system will have three sets of bused connectors called the J1, J2, and J3 backplane.
base address	A specified address that is combined with a *relative* address (or offset) to determine the *absolute* address of a data location. All VXI address windows have an associated base address for their assigned VXI address spaces.

BAV	Word Serial Byte Available command. Used to transfer 8 bits of data from a Commander to its Servant under the Word Serial Protocol.
BERR*	Bus error signal. This signal is asserted by either a slave device or the bus time out (BTO) unit when an incorrect transfer is made on the Data Transfer Bus (DTB). The BERR* signal is also used in VXI for certain protocol implementations such as writes to a full Signal register and synchronization under the Fast Handshake Word Serial Protocol.
binary	A numbering system with a base of 2.
bit	Binary digit. The smallest possible unit of data: a two-state, yes/no, 0/1 alternative. The building block of binary coding and numbering systems. Several bits make up a *byte*.
bit vector	A string of related bits in which each bit has a specific meaning.
BREQ	Word Serial Byte Request query. Used to transfer 8 bits of data from a Servant to its Commander under the Word Serial Protocol.
BTO	See *Bus Timeout Unit*.
buffer	Temporary memory/storage location for holding data before it can be transmitted elsewhere.
bus master	A device that is capable of requesting the Data Transfer Bus (DTB) for the purpose of accessing a slave device.
bus timeout unit	A VMEbus functional module that times the duration of each data transfer on the Data Transfer Bus (DTB) and terminates the DTB cycle if the duration is excessive. Without the termination capability of this module, a bus master could attempt to access a nonexistent slave, resulting in an indefinitely long wait for a slave response.
byte	A grouping of adjacent binary digits operated on by the computer as a single unit. A byte consists of 8 bits.
byte order	How bytes are arranged within a word or how words are arranged within a longword. Motorola ordering stores the most significant byte (MSB) or word first, followed by the least significant byte (LSB) or word. Intel ordering stores the LSB or word first, followed by the MSB or word.

C

clearing	Replacing the information in a register, storage location, or storage unit with zeros or blanks.
CLK10	A 10 MHz, ± 100 ppm, individually buffered (to each module slot), differential ECL system clock that is sourced from Slot 0 and distributed to Slots 1 through 12 on P2. It is distributed to each slot as a single-source, single-destination signal with a matched delay of under 8 ns.
command	A directive to a device. In VXI, three types of commands are as follows: • In Word Serial Protocol, a 16-bit imperative to a servant from its Commander (written to the Data Low register) • In Shared Memory Protocol, a 16-bit imperative from a client to a server, or vice versa (written to the Signal register) • In Instrument devices, an ASCII-coded, multi-byte directive
Commander	A message-based device which is also a bus master and can control one or more Servants.
communications registers	In message-based devices, a set of registers that are accessible to the device's Commander and are used for performing Word Serial Protocol communications.
configuration registers	A set of registers through which the system can identify a module device type, model, manufacturer, address space, and memory requirements. In order to support automatic system and memory configuration, the VXIbus specification requires that all VXIbus devices have a set of such registers.
controller	A device that is capable of controlling other devices. A desktop computer with a MXI interface board, an embedded computer in a VXI chassis, a VXI-MXI, and a VME-MXI may all be controllers depending on the configuration of the VXI system.
CR	Carriage Return; the ASCII character 0Dh.

D

Data Transfer Bus	One of four buses on the VMEbus backplane. The DTB is used by a bus master to transfer binary data between itself and a slave device.
decimal	Numbering system based upon the 10 digits 0 to 9. Also known as base 10.
de-referencing	Accessing the contents of the address location pointed to by a pointer.
default handler	Automatically installed at startup to handle associated interrupt conditions; the software can then replace it with a specified handler.
DIR	Data In Ready. This is a bit in the Response register of a message-based device that indicates that the device is ready to accept data from its Commander.
DIRviol	Data In Ready violation. A type of word serial protocol error that occurs when the Commander attempts to write data to the device when the device is not ready.
DOR	Data Out Ready. This is a bit in the Response register of a message-based device that indicates that the device is ready to output data to its Commander.
DORviol	Data Out Ready violation. A type of word serial protocol error that occurs when the Commander attempts to read data from the device when the device is not ready.
DRAM	Dynamic RAM (Random Access Memory); storage that the computer must refresh at frequent intervals.
DTB	See *Data Transfer Bus*.

E

ECL	Emitter-Coupled Logic
embedded controller	A computer plugged directly into the VXI backplane. An example is the National Instruments VXIpc-850.
END	Signals the end of a data string.

EOS	End Of String; a character sent to designate the last byte of a data message.
ERR	Protocol error
Event signal	A 16-bit value written to a message-based device's Signal register in which the most significant bit (bit 15) is a 1, designating an Event (as opposed to a Response signal). The VXI specification reserves half of the Event values for definition by the VXI Consortium. The other half are user defined.
Extended Class device	A class of VXIbus device defined for future expansion of the VXIbus specification. These devices have a subclass register within their configuration space that defines the type of extended device.
Extended Longword Serial Protocol	A form of Word Serial communication in which Commanders and Servants communicate with 48-bit data transfers.
extended controller	The external controller plus all of the extending controllers to which it is directly connected. An example is an AT-MXI connected to a VXI-MXI.
extending controller	A mainframe extender that has additional VXIbus controller capabilities. An example is the VXI-MXI.
external controller	A desktop computer or workstation connected to the VXI system via a MXI interface board. An example is a standard personal computer with a PCI-MXI-2 installed.

F

FHS	Fast Handshake; a mode of the Word Serial Protocol which uses the VXIbus signals DTACK* and BERR* for synchronization instead of the Response register bits.
FIFO	First-In-First-Out; a method of data storage in which the first element stored is the first one retrieved.

G

GPIB	General Purpose Interface Bus; the industry-standard IEEE 488 bus.
GPIO	General-Purpose I/O, a module within the National Instruments TIC chip which is used for two purposes. First, GPIOs are used for connecting external signals to the TIC chip for routing/conditioning to the VXIbus trigger lines. Second, GPIOs are used as part of a crosspoint switch matrix.

H

handshaking	A type of protocol that makes it possible for two devices to synchronize operations.
hardware context	The hardware setting for address space, access privilege, and byte ordering.
hex	Hexadecimal; the numbering system with base 16, using the digits 0 to 9 and letters A to F.
high-level	Programming with instructions in a notation more familiar to the user than machine code. Each high-level statement corresponds to several low-level machine code instructions and is machine-independent, meaning that it is portable across many platforms.
Hz	Hertz; a measure of cycles per second.

I

IACK	Interrupt Acknowledge
IEEE	Institute of Electrical and Electronic Engineers
IEEE 1014	The VME specification. Its full title is *ANSI/IEEE 1014-1987, IEEE Standard for a Versatile Backplane Bus: VMEbus.*
IEEE 1155	The VXI specification. Its full title is *ANSI/IEEE 1155-1992, VMEbus Extensions for Instrumentation: VXIbus*.
IEEE 488	Standard 488-1978, which defines the GPIB. Its full title is *IEEE Standard Digital Interface for Programmable Instrumentation*. Also referred to as IEEE 488.1 since the adoption of IEEE 488.2.

IEEE 488.2	A supplemental standard for GPIB. Its full title is *Codes, Formats, Protocols and Common Commands*.
INT8	An 8-bit signed integer; may also be called a *char*.
INT16	A 16-bit signed integer; may also be called a *short integer* or *word*.
INT32	A 32-bit signed integer; may also be called a *long* or *longword*.
interrupt	A means for a device to notify another device that an event occurred.
interrupter	A device capable of asserting interrupts and responding to an interrupt acknowledge cycle.
interrupt handler	A functional module that detects interrupt requests generated by interrupters and performs appropriate actions.
INTX	Interrupt and Timing Extension; a daughter card option for MXI mainframe extenders that extends interrupt lines and reset signals on VME boards. On VXI boards it also extends trigger lines and the VXIbus CLK10 signal.
I/O	Input/Output; the techniques, media, or devices used to achieve communication between entities.

K

KB	1,024 or 2^{10} bytes
kilobyte	2^{10}; abbreviated *KB*.

L

LF	Linefeed; the ASCII character 0Ah.
logical address	An 8-bit number that uniquely identifies the location of each VXIbus device's configuration registers in a system. The A16 register address of a device is C000h + Logical Address * 40h.
longword	Data type of 32-bit integers.

Longword Serial Protocol	A form of Word Serial communication in which Commanders and Servants communicate with 32-bit data transfers instead of 16-bit data transfers as in the normal Word Serial Protocol.
low-level	Programming at the system level with machine-dependent commands.

M

mapping	Establishing a range of address space for a one-to-one correspondence between each address in the window and an address in VXIbus memory.
master	A functional part of a MXI/VME/VXIbus device that initiates data transfers on the backplane. A transfer can be either a read or a write.
MB	1,048,576 or 2^{20} bytes
megabyte	2^{20} bytes; abbreviated *MB*.
Memory Class device	A VXIbus device that, in addition to configuration registers, has memory in VME A24 or A32 space that is accessible through addresses on the VME/VXI data transfer bus.
message-based device	An intelligent device that implements the defined VXIbus registers and communication protocols. These devices are able to use Word Serial Protocol to communicate with one another through communication registers.
MODID	A set of 13 signal lines on the VXI backplane that VXI systems use to identify which modules are located in which slots in the mainframe.
MQE	Multiple Query Error; a type of Word Serial Protocol error. If a Commander sends two Word Serial queries to a Servant without reading the response to the first query before sending the second query, a MQE is generated.
multitasking	The ability of a computer to perform two or more functions simultaneously without interference from one another. In operating system terms, it is the ability of the operating system to execute multiple applications/processes by time-sharing the available CPU resources.

MXIbus	Multisystem eXtension Interface Bus; a high-performance communication link that interconnects devices using round, flexible cables.

N

NI-VXI	The National Instruments bus interface software for VME/VXIbus systems.
nonprivileged access	One of the defined types of VMEbus data transfers; indicated by certain address modifier codes. Each of the defined VMEbus address spaces has a defined nonprivileged access mode.
NULL	A special value to denote that the contents (usually of a pointer) are invalid or zero.

O

octal	Numbering system with base 8, using numerals 0 to 7.

P

parse	The act of interpreting a string of data elements as a command to perform a device-specific action.
peek	To read the contents
pointer	A data structure that contains an address or other indication of storage location.
poke	To write a value
ppm	Parts per million
privileged access	See *Supervisory Access*.
propagation	Passing of signal through a computer system.
protocol	Set of rules or conventions governing the exchange of information between computer systems.

Q

query	Like command, causes a device to take some action, but requires a response containing data or other information. A command does not require a response.
queue	A group of items waiting to be acted upon by the computer. The arrangement of the items determines their processing priority. Queues are usually accessed in a FIFO fashion.

R

read	To get information from any input device or file storage media.
register	A high-speed device used in a CPU for temporary storage of small amounts of data or intermediate results during processing.
register-based device	A Servant-only device that supports only the four basic VXIbus configuration registers. Register-based devices are typically controlled by message-based devices via device-dependent register reads and writes.
remote controller	A device in the VXI system that has the capability to control the VXIbus, but has no intelligent CPU installed. An example is the VXI-MXI-2.
REQF	Request False; a VXI Event condition transferred using either VXI signals or VXI interrupts, indicating that a Servant no longer has a need for service.
REQT	Request True; a VXI Event condition transferred using either VXI signals or VXI interrupts, indicating that a Servant has a need for service.
resman	The name of the National Instruments Resource Manager application in the NI-VXI bus interface software. See *Resource Manager*.
Resource Manager	A message-based Commander located at Logical Address 0, which provides configuration management services such as address map configuration, Commander and Servant mappings, and self-test and diagnostic management.
Response signal	Used to report changes in Word Serial communication status between a Servant and its Commander.

ret	Return value.
RM	See *Resource Manager*
ROAK	Release On Acknowledge; a type of VXI interrupter which always deasserts its interrupt line in response to an IACK cycle on the VXIbus. All message-based VXI interrupters must be ROAK interrupters.
ROR	Release On Request; a type of VME bus arbitration where the current VMEbus master relinquishes control of the bus only when another bus master requests the VMEbus.
RORA	Release On Register Access; a type of VXI/VME interrupter which does not deassert its interrupt line in response to an IACK cycle on the VXIbus. A device-specific register access is required to remove the interrupt condition from the VXIbus. The VXI specification recommends that VXI interrupters be only ROAK interrupters.
RR	Read Ready; a bit in the Response register of a message-based device used in Word Serial Protocol indicating that a response to a previously sent query is pending.
RRviol	Read Ready protocol violation; a type of Word Serial Protocol error. If a Commander attempts to read a response from the Data Low register when the device is not Read Ready (does not have a response pending), a Read Ready violation may be generated.
rsv	Request Service; a bit in the status byte of an IEEE 488.1 and 488.2 device indicating a need for service. In VXI, whenever a new need for service arises, the rsv bit should be set and the REQT signal sent to the Commander. The rsv bit should be automatically deasserted when the Word Serial Read Status Byte query is sent.

S

s	Seconds
SEMI-SYNC Protocol	A one-line, open collector, multiple-device handshake trigger protocol.
Servant	A device controlled by a Commander.
setting	To place a binary cell into the 1 (nonzero) state.

Shared Memory Protocol	A communications protocol for message-based devices that uses a block of memory that is accessible to both a client and a server. The memory block acts as the medium for the protocol transmission.
short integer	Data type of 16 bits, same as *word*.
signal	Any communication between message-based devices consisting of a write to a Signal register. Sending a signal requires that the sending device have VMEbus master capability.
signed integer	*n* bit pattern, interpreted such that the range is from $-2^{(n-1)}$ to $+2^{(n-1)}-1$.
slave	A functional part of a MXI/VME/VXIbus device that detects data transfer cycles initiated by a VMEbus master and responds to the transfers when the address specifies one of the device's registers.
SMP	See *Shared Memory Protocol*.
SRQ	Service Request
status/ID	A value returned during an IACK cycle. In VME, usually an 8-bit value which is either a status/data value or a vector/ID value used by the processor to determine the source. In VXI, a 16-bit value used as a data; the lower 8 bits form the VXI logical address of the interrupting device and the upper 8 bits specify the reason for interrupting.
STST	START/STOP trigger protocol; a one-line, multiple-device protocol that can be sourced only by the VXI Slot 0 device and sensed by any other device on the VXI backplane.
supervisory access	One of the defined types of VMEbus data transfers; indicated by certain address modifier codes.
synchronous communications	A communications system that follows the command/response cycle model. In this model, a device issues a command to another device; the second device executes the command and then returns a response. Synchronous commands are executed in the order they are received.
SYNC Protocol	The most basic trigger protocol, simply a pulse of a minimum duration on any one of the trigger lines.
SYSFAIL*	A VMEbus signal that is used by a device to indicate an internal failure. A failed device asserts this line. In VXI, a device that fails also clears its PASSed bit in its Status register.

SYSRESET*	A VMEbus signal that is used by a device to indicate a system reset or power-up condition.
system clock driver	A VMEbus functional module that provides a 16 MHz timing signal on the utility bus.
System Controller	A functional module that has arbiter, daisy-chain driver, and MXIbus cycle timeout responsibility. Always the first device in the MXIbus daisy chain.
system hierarchy	The tree structure of the Commander/Servant relationships of all devices in the system at a given time. In the VXIbus structure, each Servant has a Commander. A Commander can in turn be a Servant to another Commander.

T

TIC	Trigger Interface Chip; a proprietary National Instruments ASIC used for direct access to the VXI trigger lines. The TIC contains a 16-bit counter, a dual 5-bit tick timer, and a full crosspoint switch.
tick	The smallest unit of time as measured by an operating system.
trigger	Either TTL or ECL lines used for intermodule communication.
tristated	Defines logic that can have one of three states: low, high, and high-impedance.
TTL	Transistor-Transistor Logic

U

UINT8	An 8-bit unsigned integer; may also be called an *unsigned char*.
UINT16	A 16-bit unsigned integer; may also be called an *unsigned short* or *word*.
UINT32	A 32-bit unsigned integer; may also be called an *unsigned long* or *longword*.
unasserted	A signal in its inactive false state.
unsigned integer	*n* bit pattern interpreted such that the range is from 0 to 2^n -1.

UnSupCom	Unsupported Command; a type of Word Serial Protocol error. If a Commander sends a command or query to a Servant which the Servant does not know how to interpret, an Unsupported Command protocol error is generated.

V

VIC	VXI Interactive Control program, a part of the NI-VXI bus interface software package. Used to program VXI devices, and develop and debug VXI application programs. Called *VICtext* when used on text-based platforms.
VME	Versa Module Eurocard or IEEE 1014
VMEbus Class device	Also called non-VXIbus or foreign devices when found in VXIbus systems. They lack the configuration registers required to make them VXIbus devices.
void	In the C language, a generic data type that can be cast to any specific data type.
VXIbus	VMEbus Extensions for Instrumentation
VXIedit	VXI Resource Editor program, a part of the NI-VXI bus interface software package. Used to configure the system, edit the manufacturer name and ID numbers, edit the model names of VXI and non-VXI devices in the system, as well as the system interrupt configuration information, and display the system configuration information generated by the Resource Manager. Called *VXItedit* when used on text-based platforms.

W

word	A data quantity consisting of 16 bits.
Word Serial Protocol	The simplest required communication protocol supported by message-based devices in the VXIbus system. It utilizes the A16 communication registers to perform 16-bit data transfers using a simple polling handshake method.

WR	Write Ready; a bit in the Response register of a message-based device used in Word Serial Protocol indicating the ability for a Servant to receive a single command/query written to its Data Low register.
write	Copying data to a storage device.
WRviol	Write Ready protocol violation; a type of Word Serial Protocol error. If a Commander attempts to write a command or query to a Servant that is not Write Ready (already has a command or query pending), a Write Ready protocol violation may be generated.
WSP	See *Word Serial Protocol*.