

ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Visit our website - *Click HERE*

NI GPIB-Serial Converter Help

2024-11-14

Contents

NI GPIB-Serial Converter Help 9
Related Documentation 9
Using Help 10
Conventions 10
Navigating Help 11
Searching Help 11
Printing Help File Topics 13
Hardware Overview 13
What You Need to Get Started 14
GPIB-RS232 Hardware Overview 14
GPIB-RS485/RS422 Hardware Overview 15
LEDs 15
RS232 Connector 16
RS485/RS422 Connector 17
GPIB Connector 17
Modes of Operation 18
Operating in S Mode 19
Operating in G Mode 20
Operating in D Mode 21
Operating in C Mode 21
Choosing Between Modes 22
Choosing Between S Mode and C Mode 22
Choosing Between G Mode and D Mode 22
Data Buffering and Serial Flow Control 23
Hardware Handshaking 23
XON/XOFF Software Handshaking 24
Serial Flow Control in S Mode 25
Serial Flow Control in G Mode 26
Serial Flow Control in D Mode 26
Serial Flow Control in C Mode 26
Getting Started with the GPIB-RS232/485/422 27
Check the Hardware Configuration 27

Install the Software 28
Connect the Hardware 29
Configure the Hardware 29
Using the NI GPIB-Serial Converter Wizard 30
Launching the NI GPIB-Serial Converter Wizard 31
Connecting the GPIB-RS232/485/422 32
No Interfaces Connected 34
Selecting an Interface 34
Locating the GPIB-RS232/485/422 35
Error Locating the GPIB-RS232/485/422 37
Incompatible Firmware Version 38
Selecting an Action 38
Using the NI GPIB-Serial Converter Wizard to Configure the GPIB-RS232/485/422 . 39
Configuration of S Mode 41
Configuration of G Mode 42
Configuration of D Mode 42
Configuration of C Mode 44
Verifying Settings 46
Applying Settings 47
Update Failed 49
Using the NI GPIB-Serial Converter Wizard to Update the GPIB-RS232/485/422 Firmware 49
Incompatible Firmware Image 50
File Error 51
Confirm Firmware 52
Updating Firmware 53
Update Failed 55
Using the GPIB-RS232/485/422 After Running the NI GPIB-Serial Converter Wizard 56
Programming Background and Considerations for the GPIB-RS232/485/422 56
Programming Steps and Examples 57
General Programming Steps for Communicating with the GPIB-RS232/485/422 Over Serial 57
Serial Initialization Code Using VISA 57
General Programming Steps for Communicating with the GPIB-RS232/485/422

Over GPIB . . . 58
GPIB Initialization Code Using VISA . . . 59
Programming in S Mode . . . 59
Status Information and Error Handling Characteristics . . . 61
Programming Considerations . . . 61
Programming Messages . . . 62
Programming Message Format . . . 62
Programming Message Example . . . 62
Programming Message Example with Data String . . . 62
How Messages Are Processed . . . 63
Function Arguments . . . 63
Abbreviations for Arguments . . . 63
GPIB Address . . . 64
Lists of GPIB Addresses . . . 64
Numeric String Arguments . . . 64
GPIB Read and Write Termination Methods (END and EOS) . . . 65
Function Names . . . 65
S Mode Default Settings and Related Functions . . . 65
List of S Mode Functions by Group . . . 66
GPIB Functions . . . 67
Serial Port Functions . . . 68
General-Use Functions . . . 68
Alphabetical List of S Mode Functions . . . 69
S Mode Functions . . . 70
cac . . . 70
caddr . . . 72
clr . . . 74
cmd . . . 75
conf . . . 77
echo . . . 79
eos . . . 80
eot . . . 83
gts . . . 85
id . . . 87
ist . . . 89
lines . . . 90

ln . . . 92
loc . . . 94
onl . . . 96
pct . . . 97
ppc . . . 98
ppu . . . 100
rd . . . 102
rpp . . . 105
rsc . . . 106
rsp . . . 108
rsv . . . 110
sic . . . 111
spign . . . 113
sre . . . 114
stat . . . 116
tmo . . . 125
trg . . . 127
wait . . . 128
wrt . . . 132
xon . . . 134
Programming in G Mode . . . 136
Status Information and Error Handling Characteristics . . . 138
Programming Considerations . . . 138
Programming Messages . . . 139
Programming Message Format . . . 139
Programming Message Example 1 . . . 139
Programming Message Example 2 . . . 140
How Messages Are Processed . . . 140
Function Arguments . . . 140
Abbreviations for Arguments . . . 141
Addressing the GPIB-RS232/485/422 and Serial Device . . . 141
Address of the GPIB-RS232/485/422 . . . 141
Address of the Serial Device . . . 141
Addressing the GPIB-RS232/485/422 and Serial Device as Listeners . . . 141
Addressing the GPIB-RS232/485/422 and Serial Device as Talkers . . . 142
GPIB Read and Write Termination Methods (END and EOS) . . . 143

Serial Port Transmission 144
Operation of the GPIB-RS232/485/422 as a GPIB Device 144
Serial Poll Responses 144
Service Request Conditions 145
Parallel Polls 145
Take Control (TCT) 146
Group Execute Trigger (GET) 146
Go To Local (GTL) 146
Device Clear 146
Function Names 146
G Mode Default Settings and Related Functions 147
List of G Mode Functions by Group 147
GPIB Functions 148
Serial Port Functions 148
General-Use Functions 148
Alphabetical List of G Mode Functions 148
G Mode Functions 149
echo 149
eos 151
id 153
onl 154
spign 156
spset 157
srqen 159
stat 160
xon 166
D Mode Overview 168
Data Transfers in D Mode 169
Using the SRQ Feature 169
Conditions that Cause SRQ to Be Asserted 170
Maximizing Performance with SRQ 170
GPIB-RS232/485/422 Serial Poll Responses 171
GPIB-RS232/485/422 Operation in Listen-Only Mode 171
C Mode Overview 172
Data Transfers in C Mode 172
Modes of Operation Within C Mode 173

Timeout Termination Mode 173
<CR> or <LF> Termination Mode 174
No-Swap Mode 175
Status and Error Message Information 176
Status Bits 176
GPIB Error Codes 178
Serial Port Error Codes 181
Interfacing to an RS232 Device 182
RS232 Standard 183
Description of the RS232 Port 184
DTE vs. DCE 184
Interfacing Serial Devices to the RS232 Serial Port. 185
Interfacing the GPIB-RS232 to a DCE with Handshaking. 186
Interfacing the GPIB-RS232 to a DCE without Handshaking 187
Minimum Configuration Cable 188
Custom Cables 188
Interfacing the GPIB-RS232 to a DTE with Handshaking 189
Interfacing the GPIB-RS232 to a DTE without Handshaking. 190
Minimum Configuration Cable 191
Custom Cables 192
Interfacing to an RS485 Device 192
RS422 193
GPIB Basics 193
Types of GPIB Messages 194
Parallel Polling 194
Operation. 195
Configuration 195
Issuing Remote Configurations in S Mode 197
Issuing Local Configurations in S Mode. 198
The Parallel Poll 198
S Mode Parallel Polling Examples. 199
Example 1. 199
Example 2. 201
Specifications 202
Multiline Interface Messages 205
Troubleshooting. 211

Differences Between the GPIB-232/485CT/CV-A and GPIB-RS232/485/422. 211
Physical Dimensions 216
Glossary 216
Important Information 221
Warranty. 221
Copyright 222
Trademarks 223
Patents 223
Warning Regarding the Use of NI Products. 224
Technical Support and Professional Services 225

NI GPIB-Serial Converter Help

September 2006, 371951A-01

The ***NI GPIB-Serial Converter Help*** describes the features, functions, and operation of the GPIB-RS232/485/422. This help assumes you have a general knowledge of RS232, RS485, or RS422 serial communications and the GPIB.

For more information about this help file, refer to the following topics:

Using Help

Related Documentation

Glossary

Important Information

Technical Support and Professional Services

Related Documentation

The following documents contain information that you may find helpful as you use this help file:

- ***ANSI/EIA-232-D, Interface Between Data Terminal Equipment and Data Circuit-Terminating Equipment Employing Serial Binary Data Interchange***
- ***EIA-485, Standard for Electrical Characteristics of Generators and Receivers for Use in Balanced Digital Multipoint Systems***

- ***EIA-422-A, Electrical Characteristics of Balanced Voltage Digital Interface Circuits***
- ***ANSI/IEEE Standard 488.1-1987, IEEE Standard Digital Interface for Programmable Instrumentation***
- ***ANSI/IEEE Standard 488.2-1992, IEEE Standard Codes, Formats, Protocols, and Common Commands***

Using Help

Conventions

Navigating Help

Searching Help

Printing Help File Topics

Conventions

This help file uses the following formatting and typographical conventions:

»	The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box.
	This icon denotes a note, which alerts you to important information.
	This icon denotes a caution, which advises you of precautions to take to avoid injury, data loss, or a system crash.
bold	Bold text denotes items that you must select or click in the software, such as menu items and dialog box options. Bold text also denotes parameter names.
dark red	Text in this color denotes a caution.
GPIB-RS232/485/422	GPIB-RS232/485/422 refers to either the GPIB-RS232 or GPIB-RS485/RS422 box.
green	Underlined text in this color denotes a link to a help topic, help file, or Web address.

IEEE 488 and IEEE 488.2	IEEE 488 and IEEE 488.2 refer to the ANSI/IEEE Standard 488.1-1987 IEEE 488.2 and ANSI/IEEE Standard 488.2-1992, respectively, which define the GPIB.
italic	Italic text denotes variables, emphasis, cross–references, or an introduction to a key concept. Italic text also denotes text that is a placeholder for a word or value that you must supply.
`monospace`	Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames, and extensions.
monospace bold	Bold text in this font denotes the messages and responses that the computer automatically prints to the screen. This font also emphasizes lines of code that are different from the other examples.
RS232	RS232 refers to the ANSI/EIA-232-D standard.
RS422	RS422 refers to the EIA-422-A standard.
RS485	RS485 refers to the EIA-485 standard.

Navigating Help (Windows Only)

To navigate this help file, use the **Contents**, **Index**, and **Search** tabs to the left of this window or use the following toolbar buttons located above the tabs:

- **Hide**—Hides the navigation pane from view.
- **Locate**—Locates the currently displayed topic in the **Contents** tab, allowing you to view related topics.
- **Back**—Displays the previously viewed topic.
- **Forward**—Displays the topic you viewed before clicking the **Back** button.
- **Options**—Displays a list of commands and viewing options for the help file.

Searching Help (Windows Only)

Use the **Search** tab to the left of this window to locate content in this help file. If you want to search for words in a certain order, such as "related documentation," add quotation marks around the search words as shown in the example. Searching for

terms on the **Search** tab allows you to quickly locate specific information and information in topics that are not included on the **Contents** tab.

Wildcards

You also can search using asterisk (*) or question mark (?) wildcards. Use the asterisk wildcard to return topics that contain a certain string. For example, a search for "prog*" lists topics that contain the words "program," "programmatically," "progress," and so on.

Use the question mark wildcard as a substitute for a single character in a search term. For example, "?ext" lists topics that contain the words "next," "text," and so on.



Note Wildcard searching will not work on Simplified Chinese, Traditional Chinese, Japanese, and Korean systems.

Nested Expressions

Use nested expressions to combine searches to further refine a search. You can use Boolean expressions and wildcards in a nested expression. For example, "example AND (program OR VI)" lists topics that contain "example program" or "example VI." You cannot nest expressions more than five levels.

Boolean Expressions

Click the ▸ button to add Boolean expressions to a search. The following Boolean operators are available:

- **AND** (default)—Returns topics that contain both search terms. You do not need to specify this operator unless you are using nested expressions.
- **OR**—Returns topics that contain either the first or second term.
- **NOT**—Returns topics that contain the first term without the second term.
- **NEAR**—Returns topics that contain both terms within eight words of each other.

Search Options

Use the following checkboxes on the **Search** tab to customize a search:

- **Search previous results**—Narrows the results from a search that returned too many topics. You must remove the checkmark from this checkbox to search all topics.
- **Match similar words**—Broadens a search to return topics that contain words similar to the search terms. For example, a search for "program" lists topics that include the words "programs," "programming," and so on.
- **Search titles only**—Searches only in the titles of topics.

Printing Help File Topics (Windows Only)

Complete the following steps to print an entire book from the **Contents** tab:

1. Right-click the book.
2. Select **Print** from the shortcut menu to display the **Print Topics** dialog box.
3. Select the **Print the selected heading and all subtopics** option.



Note Select **Print the selected topic** if you want to print the single topic you have selected in the **Contents** tab.

4. Click the **OK** button.

Printing PDF Documents

This help file may contain links to PDF documents. To print PDF documents, click the print button located on the Adobe Acrobat Viewer toolbar.

Hardware Overview

The following topics list what you need to get started and optional equipment you can order, and briefly describe the GPIB-RS232/485/422.

What You Need to Get Started

GPIB-RS232 Hardware Overview

GPIB-RS485/RS422 Hardware Overview

LEDs

RS232 Connector

RS485 Connector

GPIB Connector

What You Need to Get Started

Before you install your NI GPIB-Serial Converter hardware, be sure you have the following items:

- One of the following controllers, which is included in your kit:
  - GPIB-RS232
  - GPIB-RS485/RS422
- A DC power supply, also included in your kit.
- A shielded serial cable. Refer to Interfacing to an RS232 Device or Interfacing to an RS485 Device for your appropriate cable requirements. To connect to the PC serial port, use one of the following PC-compatible shielded cables, which you can purchase from National Instruments:
  - RS232 DTE-to-DTE cable (1, 2, or 4 m) for the GPIB-RS232 (182238-01, 182238-02, or 182238-04)
  - RS485 null-modem cable (1, 2, or 4 m) for the GPIB-RS485/RS422 (183283-01, 183283-02, or 183283-04)
- A type X2 double-shielded GPIB cable (1, 2, or 4 m), which you can purchase from National Instruments (763061-01, 763061-02, or 763061-03).

Note To meet FCC emission limits for this device, you must operate this device with shielded cables.

GPIB-RS232 Hardware Overview

The GPIB-RS232 is a high-performance serial-to-GPIB interface. It enables a computer with an RS232 serial port to be a Talker, Listener, or Controller on the GPIB. The GPIB-RS232 can also interface RS232 instruments and peripherals to the GPIB.

The GPIB-RS232 has all the software and logic required to implement the physical and electrical specifications of the IEEE 488 and RS232 standards. It can interpret and execute high-level commands you send to it over the serial port, performing GPIB-to-RS232 protocol conversions. The GPIB-RS232 also conforms to all versions of the IEEE 488 standard, including IEEE 488.2. The TNT5004 Controller chip implements all IEEE 488 Talker/Listener/Controller functionality.

GPIB-RS485/RS422 Hardware Overview

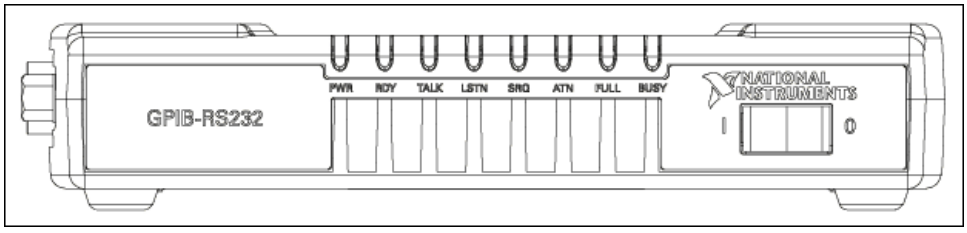
The GPIB-RS485/RS422 is a high-performance serial-to-GPIB interface. It enables a computer with an RS485 or RS422 serial port to be a Talker, Listener, or Controller on the GPIB. The GPIB-RS485/RS422 can also interface RS485 or RS422 instruments and peripherals to the GPIB.

The GPIB-RS485/RS422 does not support any multidrop protocols. The GPIB-RS485/RS422 has all the software and logic required to implement the physical and electrical specifications of the IEEE 488, RS485, and RS422 standards. It can interpret and execute high-level commands you send to it over the serial port, performing all GPIB-to-RS485 and GPIB-to-RS422 protocol conversions. The GPIB-RS485/RS422 also conforms to all versions of the IEEE 488 standard, including IEEE 488.2. The TNT5004 Controller chip implements all IEEE 488 Talker/Listener/Controller functionality.

LEDs

The eight light-emitting diodes (LEDs) are on the front of the GPIB-RS232/485/422, as shown in the following figure.

Note The figure shows only the GPIB-RS232, but the GPIB-RS485/RS422 is similar.



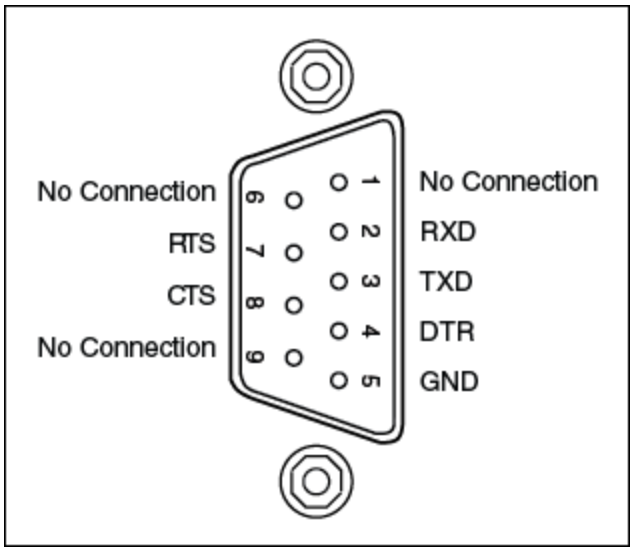
The LEDs show the GPIB-RS232/485/422 current status at all times. The following table

describes each LED.

LED	Indication
PWR	Indicates that power to the unit has been applied and the ON/OFF switch is in the ON position.
RDY	Indicates that the power-on self-test has passed successfully, and the unit is ready to operate.
TALK	Indicates that the GPIB-RS232/485/422 is addressed as a GPIB Talker.
LSTN	Indicates that the GPIB-RS232/485/422 is addressed as a GPIB Listener.
SRQ	Indicates that the GPIB signal line SRQ* is asserted.
ATN	Indicates that the GPIB signal line ATN* is asserted.
FULL	Indicates that one of the GPIB-RS232/485/422 data buffers is full. This is not an error condition, but is merely a signal that bus performance may be reduced to the speed of the slower port.
BUSY	Does not apply in S Mode. In G and D Modes, indicates there is currently GPIB data buffered in the GPIB-RS232/485/422 to be transferred over serial. In C Mode, indicates there is currently serial data buffered in the GPIB-RS232/485/422 to be transferred over GPIB.**
* Indicates that the signal is active low (negative logic or low when asserted).	
** The behavior of this LED is slightly different between the GPIB-RS232/485/422 and GPIB-232/485/CT/CV-A. For more information, refer to Differences Between the GPIB-232/485CT/CV-A and GPIB-RS232/485/422.	

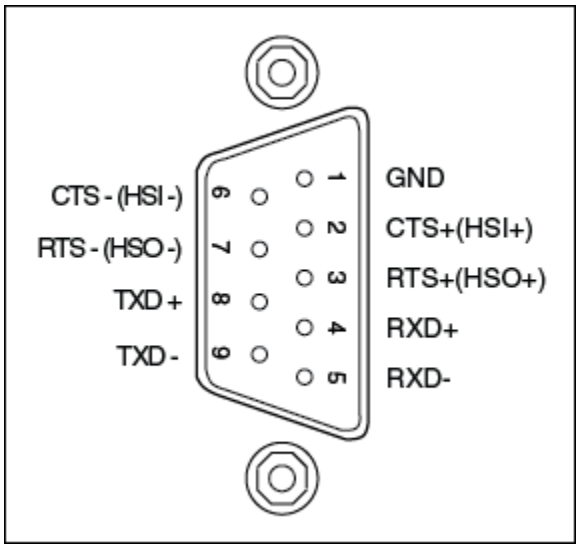
RS232 Connector

The RS232 port on the GPIB-RS232 is configured as a Data Terminal Equipment (DTE) and uses a standard 9-pin shielded D-Subminiature male connector with screwlock assemblies. The RS232 connector accepts standard 9-pin D-Subminiature female connectors. The following figure shows the RS232 connector and the signals supported. For more information about the RS232 signals, refer to Interfacing to an RS232 Device.



RS485/RS422 Connector

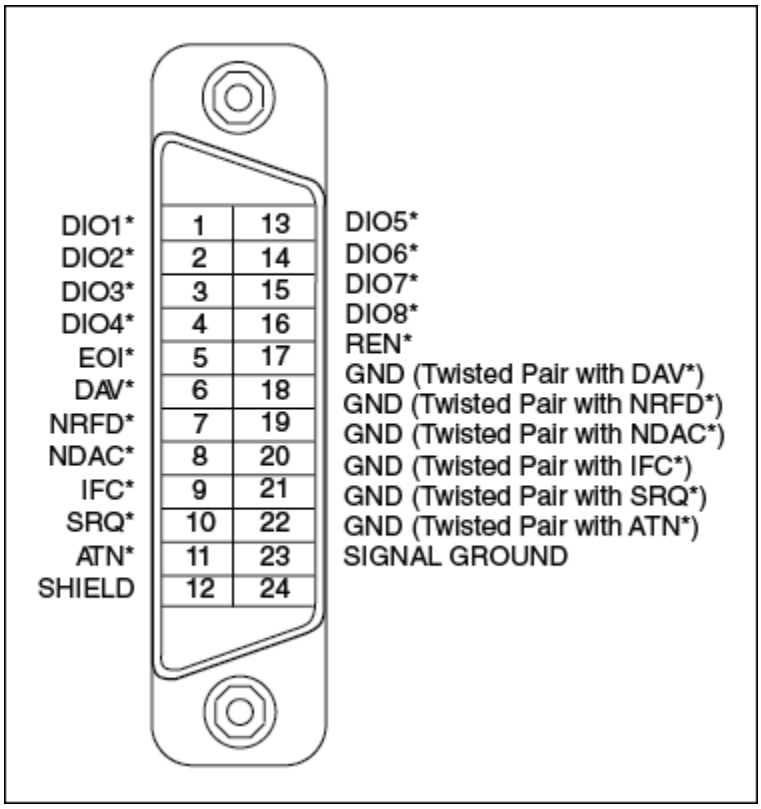
The RS485 port on the GPIB-RS485/RS422 uses a standard 9-pin shielded D-Subminiature male connector with screwlock assemblies. The RS485 connector accepts standard 9-pin D-Subminiature female connectors. The following figure shows the serial connector and the signals supported. For more information about the RS485 signals, refer to Interfacing to an RS485 Device.



GPIB Connector

The GPIB connector is a standard 24-pin shielded Champ female connector with metric

screwlock hardware. The following figure shows the GPIB connector and the signals supported. A * suffix indicates the signal is active low. Refer to GPIB Basics for more information about the GPIB signal lines.



Modes of Operation

The following topics describe the four GPIB-RS232/485/422 modes of operation and help you determine which mode of operation to use. The four modes are:

- S Mode
- G Mode
- D Mode
- C Mode

Operating in S Mode

Operating in G Mode

Operating in D Mode

Operating in C Mode

Choosing Between Modes

Choosing between S Mode and C Mode

Choosing between G Mode and D Mode

Data Buffering and Serial Flow Control

Hardware Handshaking

XON/XOFF Software Handshaking

Serial Flow Control in S Mode

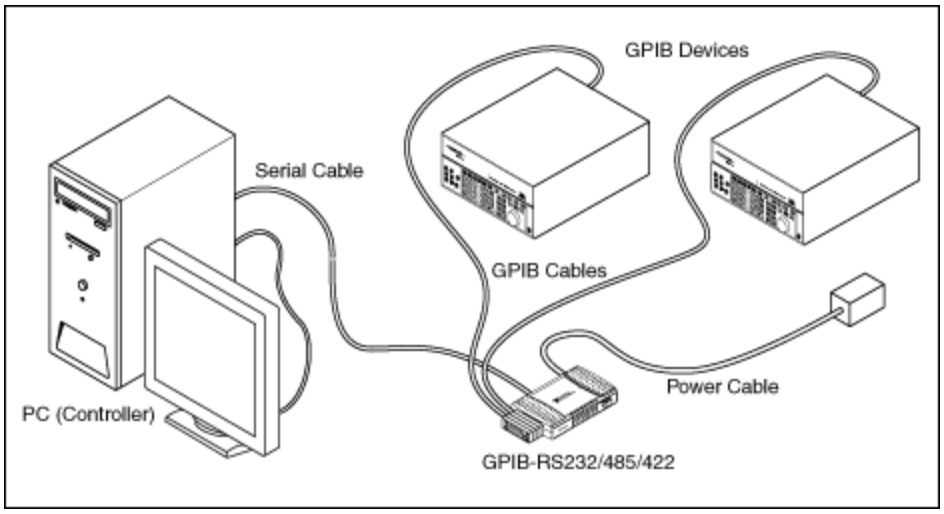
Serial Flow Control in G Mode

Serial Flow Control in D Mode

Serial Flow Control in C Mode

Operating in S Mode

Configure the GPIB-RS232/485/422 to operate in S Mode if your serial device acts as a Controller in the GPIB system, addressing one or more devices, and performing other GPIB Controller functions.

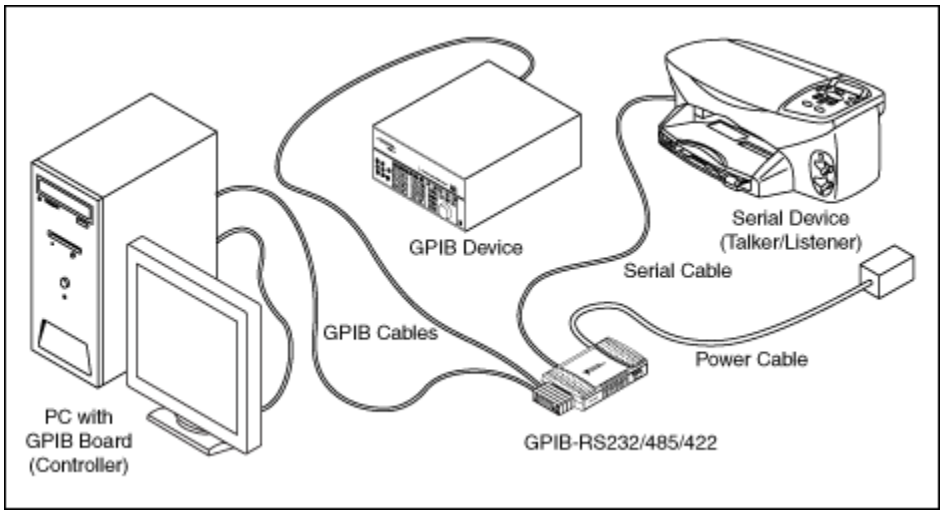


Refer to Using the NI GPIB-Serial Converter Wizard to Configure the GPIB-RS232/485/422 for detailed information about setting up your GPIB-RS232/485/422 to operate in S Mode. Refer to Programming in S Mode and S Mode Functions for information about programming the GPIB-RS232/485/422 in S Mode.

Operating in G Mode

Configure the GPIB-RS232/485/422 to operate in G Mode if your serial device acts only as a Talker and/or Listener while a GPIB controller manages the system, sending and receiving data to and from the serial device.

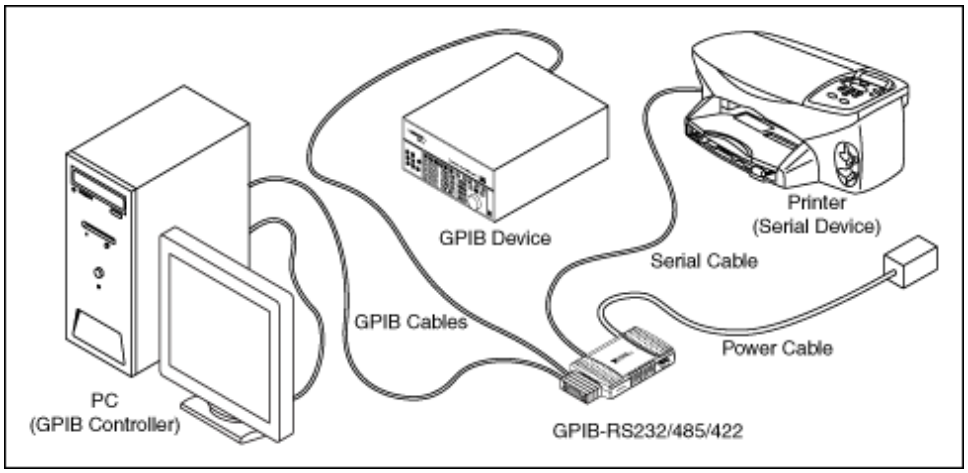
The following figure shows an example of a setup using G Mode. The GPIB-RS232/485/422 is connected to a serial printer, which the GPIB Controller programs.



Refer to Using the NI GPIB-Serial Converter Wizard to Configure the GPIB-RS232/485/422 for detailed information about setting up your GPIB-RS232/485/422 to operate in G Mode. Refer to Programming in G Mode and G Mode Functions for information about programming in G Mode.

Operating in D Mode

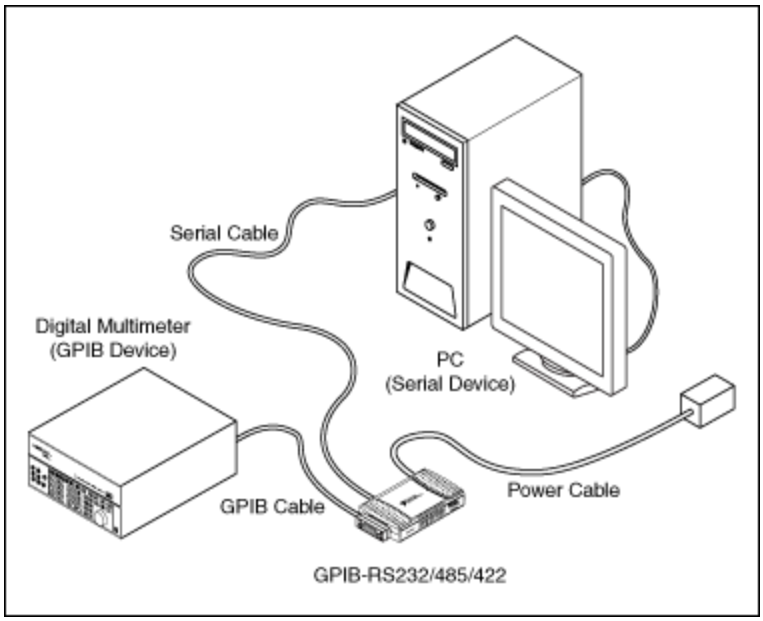
Set the GPIB-RS232/485/422 to operate in D Mode to connect a serial device to a GPIB system with a GPIB Controller responsible for addressing the GPIB-RS232/485/422 to talk or listen. The following figure shows an example of a system requiring D Mode. The system includes a serial device, such as a printer, connected to a GPIB system with an existing GPIB Controller, such as a computer with a National Instruments PCI-GPIB IEEE 488.2 board installed.



Refer to Using the NI GPIB-Serial Converter Wizard to Configure the GPIB-RS232/485/422 for detailed information about setting up your GPIB-RS232/485/422 to operate in D mode. Refer to D Mode Overview for information about using D Mode.

Operating in C Mode

Set the GPIB-RS232/485/422 to operate in C Mode when transferring data between a single GPIB device with no Controller capabilities and a serial device. The following figure shows an example of a system requiring C Mode. The system includes a GPIB device, such as a digital multimeter, connected to a computer's serial port through a GPIB-RS232/485/422.



Refer to Using the NI GPIB-Serial Converter Wizard to Configure the GPIB-RS232/485/422 for detailed information about setting up your GPIB-RS232/485/422 to operate in C mode. Refer to C Mode Overview for information about using C Mode.

Choosing Between Modes

You can connect the GPIB-RS232/485/422 to a serial device and one or more GPIB devices. The way you use the serial device in your system setup determines the mode of operation to use.

If the serial device is the controller, use either S Mode or C Mode. If the GPIB device is the controller, use either G Mode or D Mode.

Choosing Between S Mode and C Mode

Use S Mode when communicating with more than one GPIB device. You should also consider S Mode if you are using more advanced GPIB controller functionality beyond talker/listener functionality.

Use C Mode when communicating with a single GPIB device only using talker/listener functionality.

Choosing Between G Mode and D Mode

Use D Mode when sending GPIB data to a serial instrument and receiving data from a serial instrument over GPIB. In addition, you should consider D Mode when the serial settings of a serial instrument are known in advance. D Mode also offers the Listen-Only method of operation that allows a serial port to monitor all GPIB data bus activity. For more information about D Mode, refer to D Mode Overview. For more information about the Listen-Only method of operation, refer to GPIB-RS232/485/422 Operation in Listen-Only Mode.

Use G Mode for more advanced functionality. For example, G Mode allows you to change serial port settings programmatically, reset the GPIB-RS232/485/422 internal state, configure SRQ conditions, and determine how much serial data is available to read.

For more information about G Mode, refer to Programming in G Mode and G Mode Functions.

Data Buffering and Serial Flow Control

Two protection mechanisms ensure that the GPIB-RS232/485/422 does not lose incoming serial data: data buffering and handshaking.

The GPIB-RS232/485/422 has an internal RAM buffer that stores incoming serial data until it can output the data to the GPIB port. The serial RAM buffer size is 1 MB.

When the serial RAM buffer is nearly full, the GPIB-RS232/485/422 can handshake with the serial host to stop data transmission. When more space is available in the buffer because data was read from the GPIB, the GPIB-RS232/485/422 can again handshake with the serial host to start data transmission. The GPIB-RS232/485/422 can use both the XON/XOFF software handshaking and the hardware handshaking protocols.

Hardware Handshaking

The hardware handshake function is configurable using the NI GPIB-Serial Converter Wizard. When Flow Control is set to Hardware, the device enables hardware handshake during serial data transfers and uses the Request to Send (RTS on the GPIB-RS232, RTS+ and RTS– on the GPIB-RS485/RS422) and Clear to Send (CTS on the GPIB-RS232, CTS+ and CTS– on the GPIB-RS485/RS422) signal lines. When the GPIB-RS232/

485/422 is ready to accept serial data, it asserts the RTS line(s). RTS remains asserted until the GPIB-RS232/485/422 data buffer is almost full. At this point, the GPIB-RS232/485/422 unasserts the RTS line(s), signaling to the serial host that it is no longer ready to accept data. The serial host should monitor the RTS line(s) and suspend data transmission whenever RTS becomes unasserted. The GPIB-RS232/485/422 asserts RTS when it is again ready to receive serial data.

The GPIB-RS232/485/422 can also suspend transmission when the serial device is no longer ready to accept data. The GPIB-RS232/485/422 is configured to immediately stop transmission of serial data when CTS becomes unasserted. It resumes transmission when CTS is reasserted.

Because most serial devices use the same form of hardware handshaking as the GPIB-RS232/485/422, you can achieve bidirectional flow control by using a serial cable that connects the GPIB-RS232/485/422 RTS signal(s) to the serial device CTS signal(s). In addition, the serial device RTS signal(s) should be connected to the GPIB-RS232/485/422 CTS signal(s). This setup allows each device to monitor the RTS signal(s) of the other device and to suspend transmission when necessary to prevent data loss. Refer to Interfacing to an RS232 Device or Interfacing to an RS485 Device for more information on wiring schemes.



Note Do not enable hardware handshaking on the GPIB-RS232/485/422 if your serial cable does not support it or you did not properly connect the RTS and CTS signals as described above. Enabling hardware handshaking in this case may prevent the GPIB-RS232/485/422 from sending serial data.



Note In configuration mode, the GPIB-RS232/485/422 enables hardware handshaking. If your cable does not support hardware handshaking, you may experience problems configuring your hardware.

XON/XOFF Software Handshaking

If your serial device does not implement or recognize the hardware handshake scheme, your cable does not support the necessary handshake lines, or your application software requires XON/XOFF handshaking, you can enable the XON/XOFF handshaking protocol by using the NI GPIB-Serial Converter Wizard or the `xon` command in S Mode and G Mode. This handshaking protocol performs the same

function as the hardware handshake, but does so by sending special control codes over the data lines instead of changing logic levels on dedicated control lines.

When you enable the XON/XOFF protocol, the GPIB-RS232/485/422 sends the XOFF character (decimal 19 or <CTRL-S>) before the internal buffer overflows. When the GPIB-RS232/485/422 can start receiving characters again, it sends the XON character (decimal 17 or <CTRL-Q>). Similarly, if the GPIB-RS232/485/422 is transmitting data and receives the XOFF character, it suspends transmission of any further data until it receives the XON character.

If you are transmitting binary data (as opposed to 7-bit ASCII), do not configure the GPIB-RS232/485/422 to use XON/XOFF software handshaking. Because the binary data could contain any binary sequence, including decimal 19 (<Ctrl-S>) or decimal 17 (<Ctrl-Q>), the GPIB-RS232/485/422 could not distinguish between data values or handshake control codes. If XON/XOFF software handshaking were enabled in this case, the GPIB-RS232/485/422 would handshake erratically.

Serial Flow Control in S Mode

Use the NI GPIB-Serial Converter Wizard to configure GPIB-RS232/485/422 serial flow control in S Mode. There are three configuration choices, as shown in the following table. You can also modify flow control with the `xon` and `onl` commands.

Power-On State (Configured Using the NI GPIB-Serial Converter Wizard)	**xon 1,1**	**xon 0,0**	**onl 1**
Hardware handshaking enabled*	Hardware handshaking enabled and XON/XOFF enabled	Hardware handshaking enabled*	Hardware handshaking enabled*
XON/XOFF enabled*	XON/XOFF enabled*	No flow control	XON/XOFF enabled
No flow control	XON/XOFF enabled*	No flow control	No flow control
* Where it does not explicitly say that hardware handshaking or XON/XOFF is enabled or disabled, it is assumed to be disabled.			

For more information about S Mode configuration, refer to Configuration of S Mode.

For more information about the `xon` and `onl` commands, refer to S Mode Functions.

Serial Flow Control in G Mode

You cannot configure GPIB-RS232/485/422 serial flow control in G Mode using the NI-Serial Converter Wizard. Instead, it is always set to be hardware handshaking enabled at power-on. However, you can modify flow control with the `xon` and `onl` commands, as shown in the following table.

Power-On State	**xon 1,1**	**xon 0,0**	**onl 1**
Hardware handshaking enabled*	Hardware handshaking disabled and XON/XOFF enabled	Hardware handshaking enabled*	Hardware handshaking enabled*
* Where it does not explicitly say that hardware handshaking or XON/XOFF is enabled or disabled, it is assumed to be disabled.			

For more information about the `xon` and `onl` commands, refer to G Mode Functions.

Serial Flow Control in D Mode

You can configure GPIB-RS232/485/422 serial flow control in D Mode only by using the NI-Serial Converter Wizard. There are three configuration choices, as shown in the following table.

Power-On State (Configured Using the NI GPIB-Serial Converter Wizard)
Hardware handshaking enabled*
XON/XOFF enabled*
No flow control
* Where it does not explicitly say that hardware handshaking or XON/XOFF is enabled or disabled, it is assumed to be disabled.

For more information about D Mode configuration, refer to Configuration of D Mode.

Serial Flow Control in C Mode

You can configure GPIB-RS232/485/422 serial flow control in C Mode only by using the NI-Serial Converter Wizard. There are three configuration choices, as shown in the

following table.

Power-On State (Configured Using the NI GPIB-Serial Converter Wizard)
Hardware handshaking enabled*
XON/XOFF enabled*
No flow control
* Where it does not explicitly say that hardware handshaking or XON/XOFF is enabled or disabled, it is assumed to be disabled.

For more information about C Mode configuration, refer to Configuration of C Mode.

Getting Started with the GPIB-RS232/485/422

The following topics contain detailed instructions for connecting and configuring your GPIB-RS232/485/422.

Check the Hardware Configuration

Install the Software

Connect the Hardware

Configure the Hardware

Check the Hardware Configuration

The GPIB-RS232/485/422 ships with a 100-240 VAC wall-mount or desktop power supply. If you plan to use a different power source to power your GPIB-RS232/485/422, verify that the voltage marked on that power supply matches the input voltage marked on your GPIB-RS232/485/422.

Caution Do not operate your GPIB-RS232/485/422 at any voltage other than that labelled on the unit. Doing so could damage the unit.

The GPIB-RS232/485/422 ships with the following factory default power-on settings:

- D Mode

- 8 data bits/character
- 1 stop bit/character
- Parity disabled
- Serial port configured to 9600 baud
- Hardware flow control
- GPIB primary address 5
- GPIB secondary address disabled
- SRQ enabled

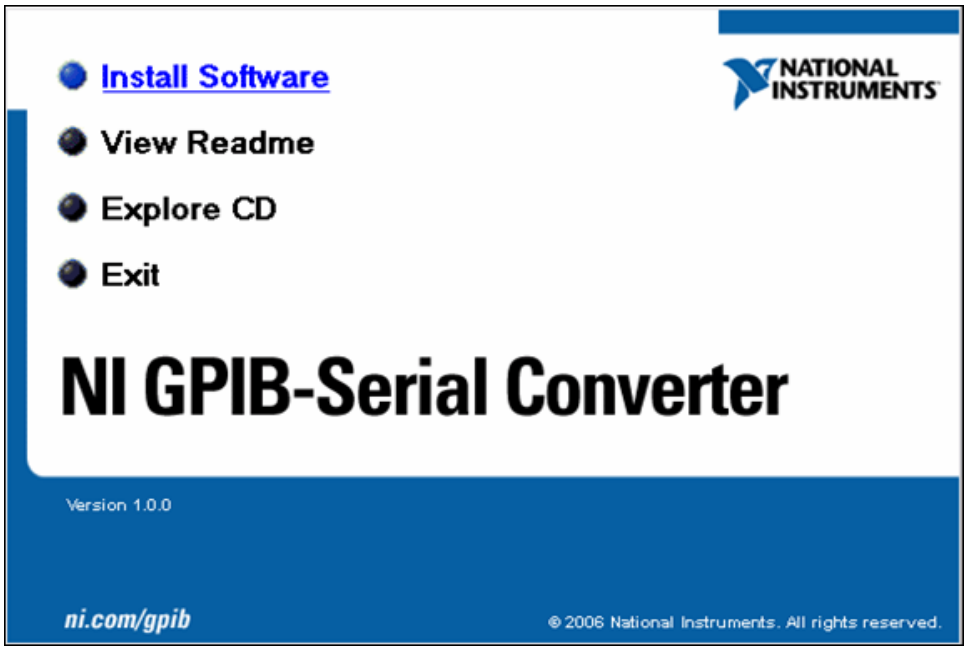
In D Mode, the GPIB-RS232/485/422 can act only as a GPIB Talker or Listener. To change any default setting, you must install the NI GPIB-Serial Converter software.

For more information about D Mode settings, refer to Configuration of D Mode. For more information about operation in D Mode, refer to D Mode Overview.

Install the Software

To install the NI GPIB-Serial Converter software, complete the following steps:

1. Log on as Administrator or as a user with administrator privileges.
2. Insert the NI GPIB-Serial Converter Software CD included with your GPIB-RS232 or GPIB-RS485/422 hardware.
3. If the following screen appears, click the **Install Software** link.



4. If the autorun application does not start, run `setup.exe` from the CD and follow the onscreen instructions to complete installation.
5. Reboot when prompted at the end of software installation.

Connect the Hardware

To connect the hardware, complete the following steps:

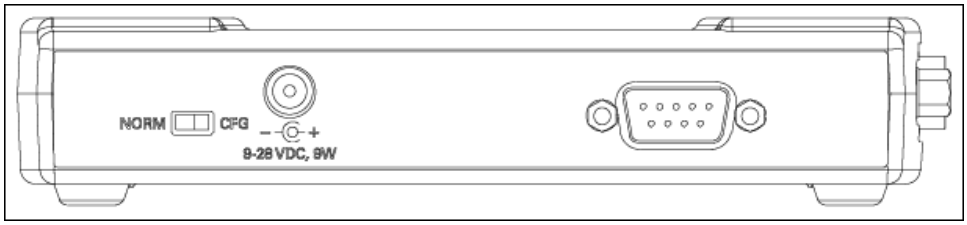
1. Connect either the GPIB or serial cable between the GPIB-RS232/485/422 and computer.
2. Connect the DC power plug of the DC power supply to the GPIB-RS232/485/422 hardware back panel and plug the power supply into an AC outlet.

Caution Do not operate your GPIB-RS232/485/422 at any voltage other than that labelled on the unit. Doing so could damage the unit.

Configure the Hardware

To configure the hardware, complete the following steps:

1. Set the configuration switch on the rear panel to **CFG**, as shown below.



Note The switch is recessed to avoid unintentional toggling during operation of the GPIB-RS232/485/422. To flip the switch, a flathead screwdriver or similar tool may be required.

2. Power on the GPIB-RS232/485/RS422 using the power switch on the front of the unit.

   The **PWR** LED indicator should come on immediately. The **RDY** LED indicator blinks green after the hardware has passed its power-on self test, indicating it is ready for configuration. For more information, refer to LEDs.

3. Launch the NI GPIB-Serial Converter Wizard from **Start»Programs»National Instruments»GPIB-Serial Converter**.
4. Follow the onscreen instructions in the wizard.

Using the NI GPIB-Serial Converter Wizard

The NI GPIB-Serial Converter Wizard allows you to modify the GPIB-RS232/485/422 configuration or update the GPIB-RS232/485/422 firmware. Prior to modifying the configuration or updating the firmware, the NI GPIB-Serial Converter Wizard must locate your GPIB-RS232/485/422. The following topics explain basic operation of the NI GPIB-Serial Converter Wizard and locating the GPIB-RS232/485/422.

Prior to launching the NI GPIB-Serial Converter Wizard, be sure the GPIB-Serial Software is installed on your computer. Refer to Getting Started with the GPIB-RS232/485/422 for more details.

Launching the NI GPIB-Serial Converter Wizard

Connecting the GPIB-RS232/485/422

No Interfaces Connected

Selecting an Interface

Locating the GPIB-RS232/485/422

Error Locating the GPIB-RS232/485/422

Incompatible Firmware Version

Selecting an Action

Using the NI GPIB-Serial Converter Wizard to Configure the GPIB-RS232/485/422

Configuration of S Mode

Configuration of G Mode

Configuration of D Mode

Configuration of C Mode

Verifying Settings

Applying Settings

Update Failed

Using the NI GPIB-Serial Converter Wizard to Update the GPIB-RS232/485/422 Firmware

Incompatible Firmware Image

File Error

Confirm Firmware

Updating Firmware

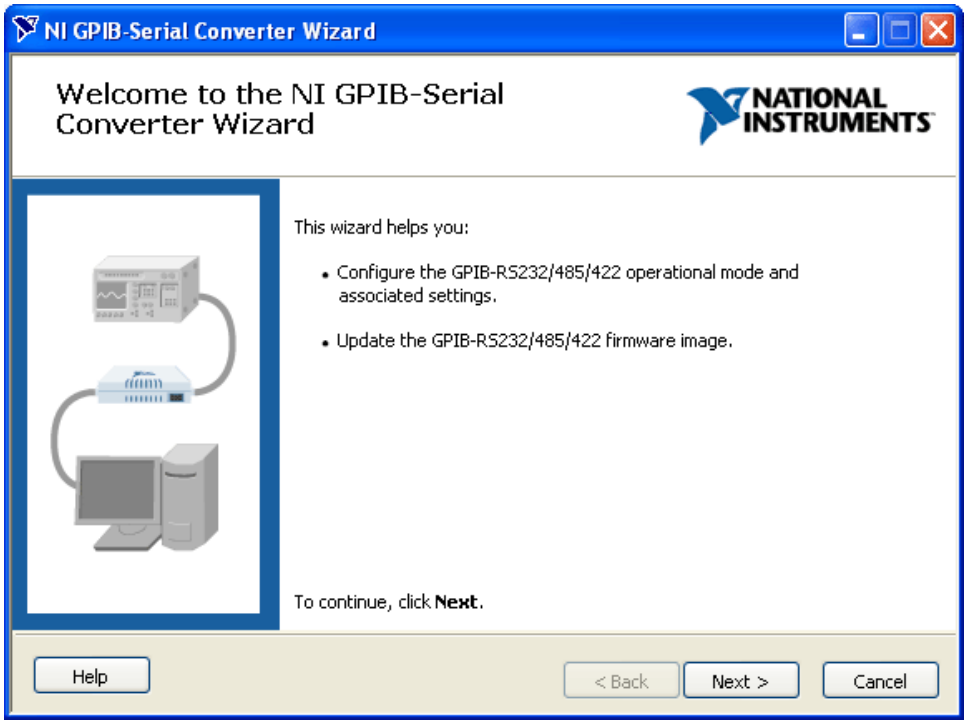
Update Failed

Using the GPIB-RS232/485/422 After Running the NI GPIB-Serial Converter Wizard

Launching the NI GPIB-Serial Converter Wizard

To run the NI GPIB-Serial Converter Wizard, go to **Start»Programs»National Instruments»GPIB-Serial Converter**.

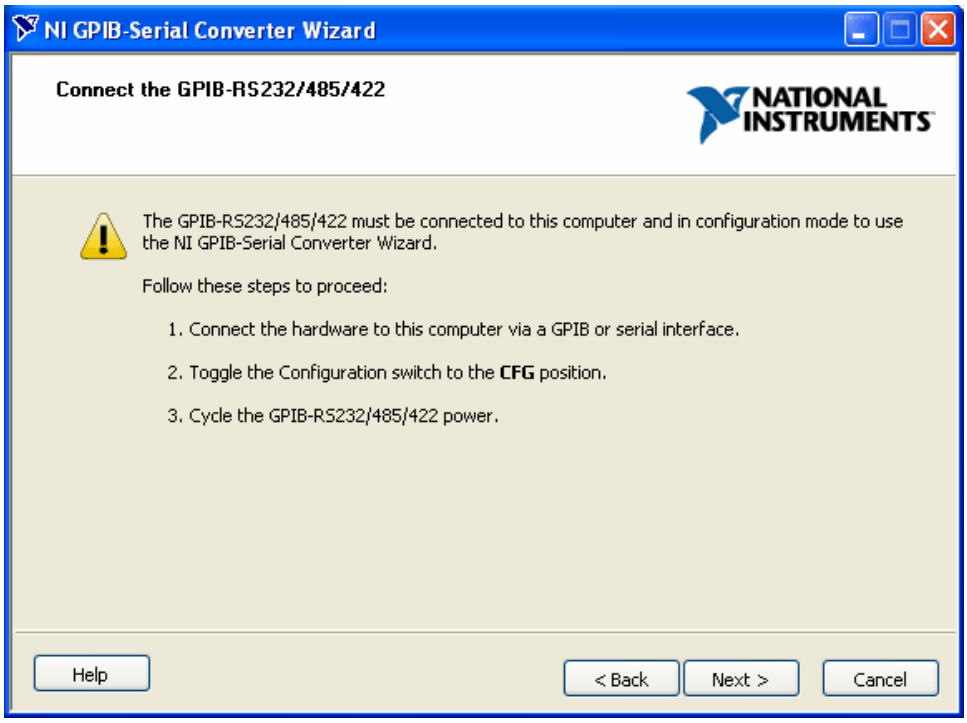
The NI GPIB-Serial Converter Wizard welcome screen should appear, as shown in the following figure.



Click **Next** to continue and proceed to Connecting the GPIB-RS232/485/422.

Connecting the GPIB-RS232/485/422

The NI GPIB-Serial Converter Wizard **Connect the GPIB-RS232/485/422** screen displays, as shown in the following figure.



You must now connect the GPIB-RS232/485/422 to your computer and set the GPIB-RS232/485/422 for Configuration mode.

You can connect the GPIB-RS232/485/422 via either GPIB or serial cable. If your computer has a GPIB interface, use a GPIB cable to connect your computer to your GPIB-RS232/485/422. If your computer does not have a GPIB interface, use a NULL modem serial cable to connect the GPIB-RS232/485/422 to your computer.

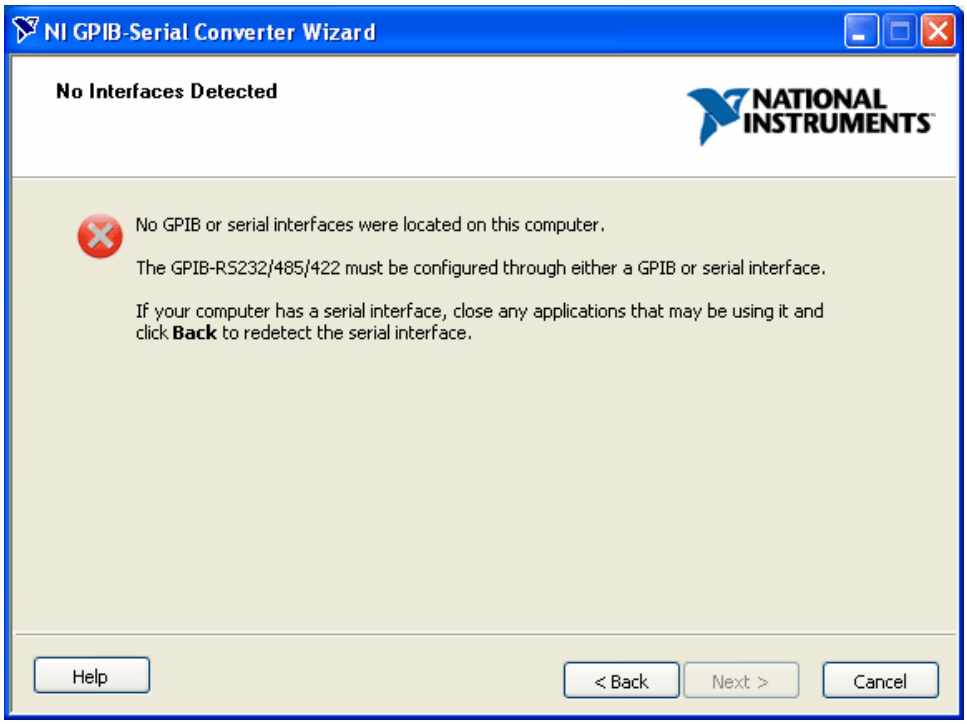
The GPIB-RS232/485/422 must now be set to run in Configuration mode. Follow these steps to run your GPIB-RS232/485/422 in Configuration mode:

1. Set the configuration switch on the GPIB-RS232/485/422 rear panel to **CFG**. Refer to Configure the Hardware for the configuration switch location.
2. Turn the power switch to **OFF** to turn off the GPIB-RS232/485/422. Refer to LEDs for the power switch location.
3. Turn the power switch to **ON** to restart the GPIB-RS232/485/422.

The GPIB-RS232/485/422 is now in Configuration mode. Click **Next** to to have the NI GPIB-Serial Converter Wizard detect serial and GPIB interfaces installed in your computer.

If at least one serial or GPIB interface is detected, proceed to Selecting an Interface.

If the wizard cannot find any serial or GPIB interfaces on your computer, the **No Interfaces Detected** screen displays, as shown in the following figure.



If the wizard appears as shown above, proceed to No Interfaces Connected.

No Interfaces Connected

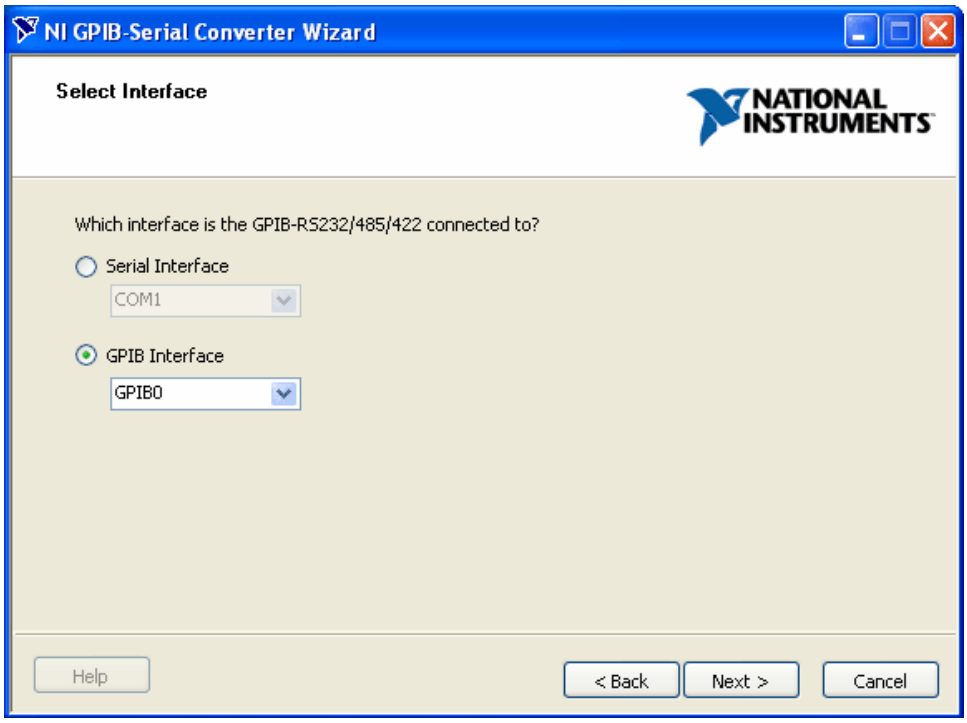
The NI GPIB-Serial Converter Wizard requires at least one serial port or one GPIB interface installed in the computer to configure your hardware.

If a serial port or GPIB interface is installed in your computer, ensure that the driver for your hardware is correctly installed and functioning properly. Verify that any other applications that may be using your interface have exited.

After exiting all applications, click **Back** to redetect your computer interfaces.

Selecting an Interface

After detecting the interfaces present on your computer, the NI GPIB-Serial Converter Wizard **Select Interface** screen should display, as shown in the following figure.

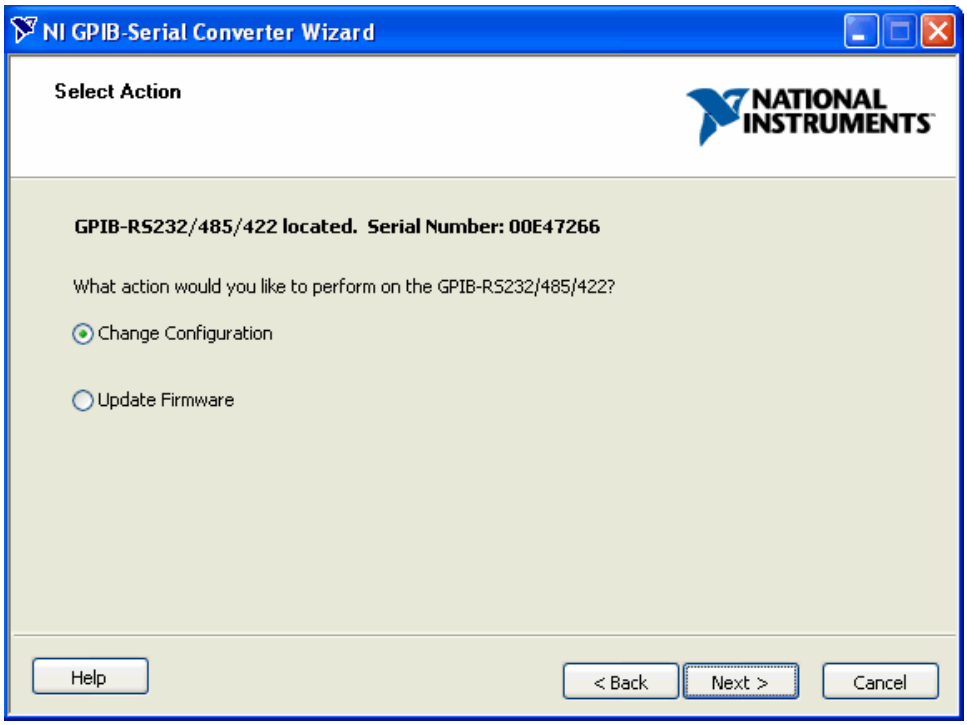


The NI GPIB-Serial Converter Wizard must know which interface you connected to the GPIB-RS232/485/422. First, select whether the GPIB-RS232/485/422 is connected to a serial or GPIB interface by clicking **Serial Interface** if connected to a serial interface or **GPIB Interface** if connected to a GPIB interface. Next, select the specific interface connected to the GPIB-RS232/485/422 from the drop-down list. For example, if the GPIB-RS232/485/422 is connected to COM1 (serial interface 1), select **COM1** from the drop-down list.

Click **Next** to allow the NI GPIB-Serial Converter Wizard to detect the GPIB-RS232/485/422 and its current configuration and proceed to Locating the GPIB-RS232/485/422.

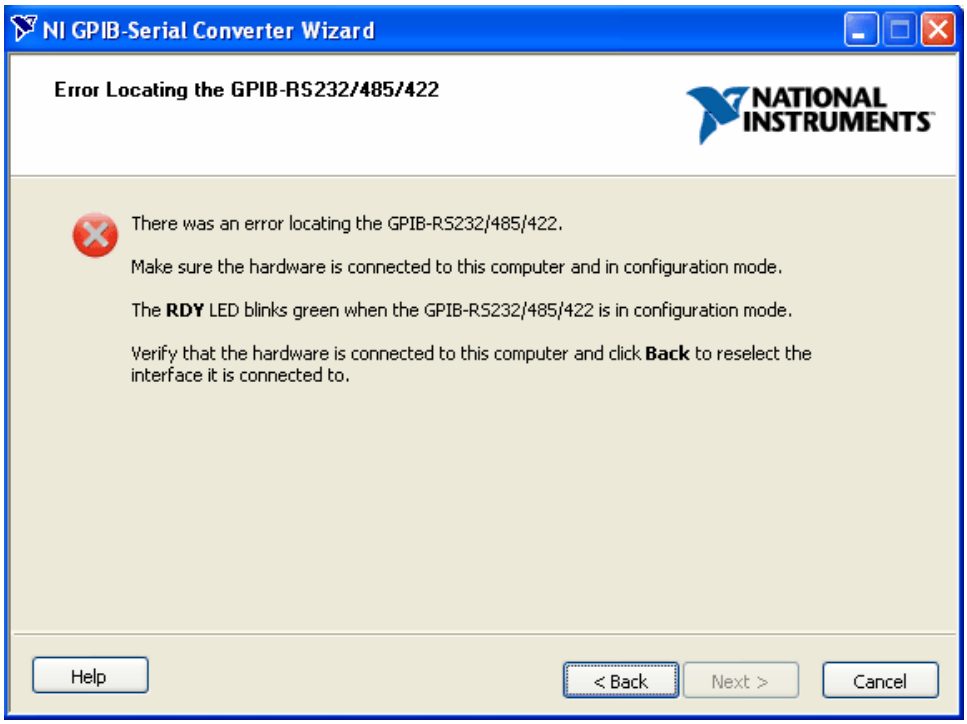
Locating the GPIB-RS232/485/422

The NI GPIB-Serial Converter Wizard attempts to locate the GPIB-RS232/485/422 attached to the interface chosen in the previous step. If successful, the **Select Action** screen appears, and your GPIB-RS232/485/422 serial number is displayed, as shown in the following figure.



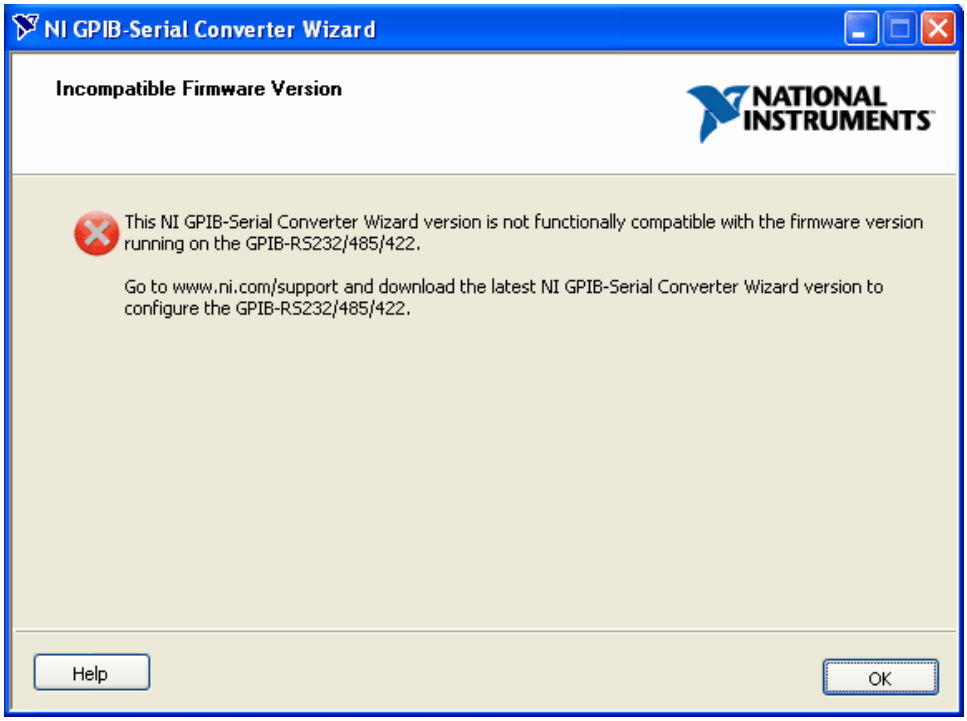
If the NI GPIB-Serial Converter Wizard displays the **Select Action** screen, proceed to Selecting an Action.

If there was an error locating the GPIB-RS232/485/422, the **Error Locating the GPIB-RS232/485/422** screen is displayed, as shown in the following figure.



If the NI GPIB-Serial Converter Wizard appears as shown above, proceed to Error Locating the GPIB-RS232/485/422.

If there is an incompatibility with the NI GPIB-Serial Converter Wizard and the firmware running on the GPIB-RS232/485/422, the **Incompatible Firmware Version** screen is displayed, as shown in the following figure.



If the NI GPIB-Serial Converter Wizard appears as shown above, proceed to Incompatible Firmware Version.

Error Locating the GPIB-RS232/485/422

If the NI GPIB-Serial Converter Wizard cannot locate the GPIB-RS232/485/422, follow these troubleshooting steps:

1. Verify that the GPIB-RS232/485/422 is connected to the computer interface specified in Selecting an Interface. Refer to Selecting an Interface for more details.
2. Verify that the GPIB-RS232/485/422 is in Configuration mode. The **RDY** LED is blinking green if the GPIB-RS232/485/422 is in Configuration mode. Refer to Connecting the GPIB-RS232/485/422 for instructions on enabling Configuration mode.
3. Verify that the cable connecting the computer to the GPIB-RS232/485/422 is

connected correctly.

4. If connecting to the GPIB-RS232/485/422 over GPIB, verify that the GPIB interface being used to configure the GPIB-RS232/485/422 is set to System Controller, is Controller In Charge of the GPIB bus, and has a Primary Address (PAD) other than 13.
5. If connecting to the GPIB-RS232/485/422 over GPIB, remove all other devices from the GPIB bus other than your computer interface card and the GPIB-RS232/485/422.
6. If connecting to the GPIB-RS232/485/422 over serial, make sure your cable support hardware handshaking and the RTS and CTS lines are connected appropriately. For more information about hardware handshaking, refer to Hardware Handshaking.

Incompatible Firmware Version

The NI GPIB-Serial Converter Wizard is not compatible with a newer version of the firmware running on the GPIB-RS232/485/422. Follow these steps to install a newer version of the wizard:

1. Exit the NI GPIB-Serial Converter Wizard.
2. Go to `www.ni.com/support`.
3. Locate and download the current version of the NI GPIB-Serial Converter Wizard.
4. Refer to Install the Software to install the new version of the NI GPIB-Serial Converter Wizard.
5. Refer to Launching the NI GPIB-Serial Converter Wizard to run the new version of the NI GPIB-Serial Converter Wizard.

Selecting an Action

The NI GPIB-Serial Converter Wizard has now detected your GPIB-RS232/485/422 and read its current settings. The wizard now can either modify the GPIB-RS232/485/422 configuration or update the GPIB-RS232/485/422 firmware.

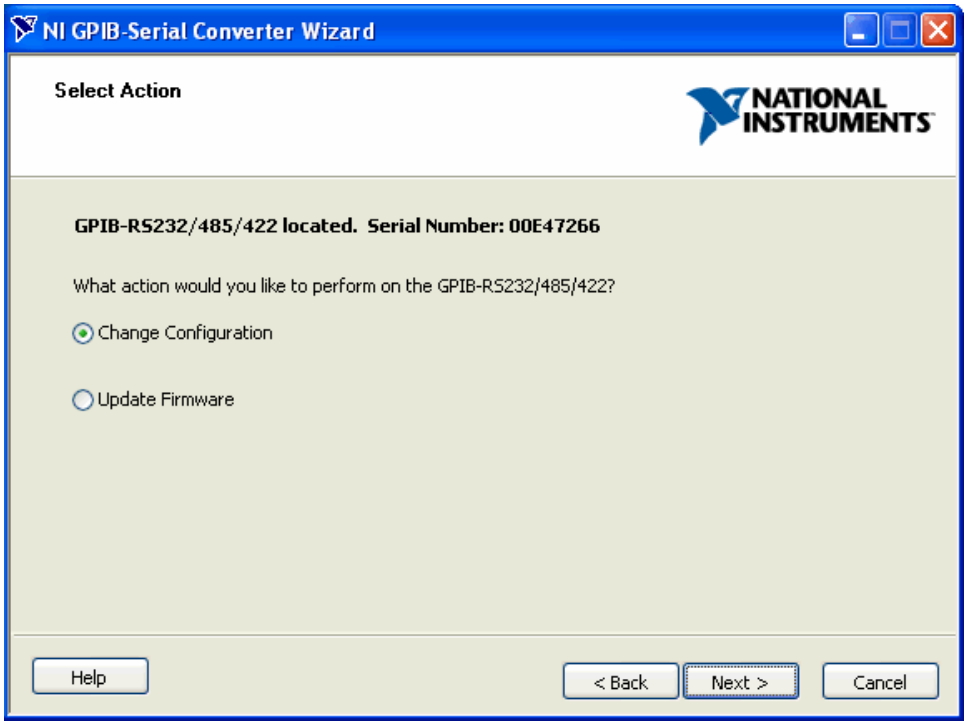
To proceed with modifying the GPIB-RS232/485/422 configuration, click **Change Configuration**. To update the GPIB-RS232/485/422 firmware, click **Update Firmware**. Click **Next** to continue with your selected action.

Using the NI GPIB-Serial Converter Wizard to Configure the GPIB-RS232/485/422

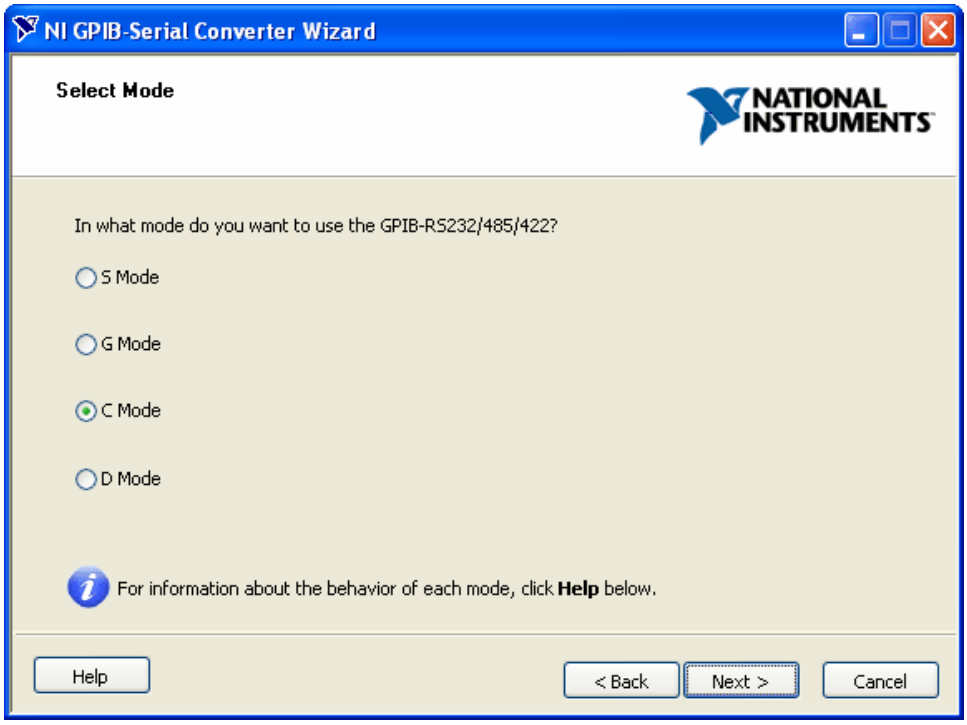
Using the NI GPIB-Serial Converter Wizard to Update the GPIB-RS232/485/422 Firmware

Using the NI GPIB-Serial Converter Wizard to Configure the GPIB-RS232/485/422

Before proceeding, the NI GPIB-Serial Converter Wizard must be running and your screen must appear as shown below. Refer to Launching the NI GPIB-Serial Converter Wizard for more details.



After you select to change the GPIB-RS232/485/422 configuration, the NI GPIB-Serial Converter Wizard **Select Mode** screen should appear as shown below.



You must choose which GPIB-RS232/485/422 mode meets your use requirements. Refer to Modes of Operation for a brief overview of the different modes and the differences between them. Refer to the following sections for complete documentation of each mode:

S Mode

G Mode

C Mode

D Mode

After choosing the mode that meets your requirements, select the mode by clicking the radio button to the left of the chosen mode. Click **Next** to proceed.

To continue with the GPIB-RS232/485/422 configuration, refer to the section associated with the mode chosen.

Configuration of S Mode

Configuration of G Mode

Configuration of D Mode

Configuration of C Mode

After selecting the desired configuration settings for the chosen mode, click **Next** to proceed to Verifying Settings.

Configuration of S Mode

When the GPIB-RS232/485/422 is configured to operate in S mode, it lets your serial device act as a Controller in the GPIB system, addressing one or more devices, and performing other GPIB Controller functions. For more details about S Mode operation, refer to Programming in S Mode.

S Mode requires the user to configure the serial settings used on the serial interface. The serial settings must exactly match the serial settings used on the host serial port. The serial settings are:

- **Baud Rate**—Number of bits transmitted over the serial interface per second (300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, and 115200).

- **Data Bits**—Number of data bits that represent a serial byte.
  - **7**—Seven data bits per data byte.
  - **8**—Eight data bits per data byte.

- **Parity**—Parity algorithm used to detect serial errors.
  - **None**—No parity is used to detect serial errors.
  - **Even**—Data bits plus the parity bit sum to an even number.
  - **Odd**—Data bits plus the parity bit sum to an odd number.

- **Stop Bits**—Number of bits per byte used to indicate the end of the serial data byte.
  - **1**—One stop bit is used for every data byte.
  - **2**—Two stop bits are used for every data byte.

- **Flow Control**—Method used to ensure serial data is not lost.
  - **Hardware**—Hardware protocol used for flow control. For more information, refer to Hardware Handshaking.
  - **XonXoff**—Software implementation of the flow control protocol. For more information, refer to XON/XOFF Software Handshaking.
  - **None**—No flow control used.

Configuration of G Mode

When the GPIB-RS232/485/422 is configured to operate in G Mode, it is used as a GPIB device sending and receiving data to and from the serial device, while a GPIB controller manages the system. GPIB-RS232/485/422 implements two GPIB devices on the same physical GPIB interface. The first address is the GPIB-RS232/485/422 address. The second address is the serial device address. Data a GPIB controller sends while the GPIB-RS232/485/422 is addressed is interpreted as programming messages. Data a GPIB controller sends while a serial device is addressed is sent over the serial interface. For more details about G Mode operation, refer to Programming in G Mode.

You can choose any valid combination of GPIB primary and secondary address for the serial device and another valid combination of GPIB primary and secondary address for the GPIB-RS232/485/422 as long as neither is used by any other GPIB device in your system. The valid addresses are:

- GPIB Primary Address: 0–30
- GPIB Secondary Address: 0 (disabled), 96–126

For descriptions of non-GPIB serial device settings that are changed programmatically by sending programming messages to the GPIB-RS232/485/422, and their power-on default values, refer to Programming in G Mode.

Configuration of D Mode

When the GPIB-RS232/485/422 is configured to operate in D mode, you can connect a serial device to a GPIB system with a GPIB Controller responsible for addressing the GPIB-RS232/485/422 to talk or listen. For more details about the D Mode operation, refer to D Mode Overview.

D Mode requires the user to configure the serial settings used on the serial interface, as well as the GPIB settings used on the GPIB interface. Below is a brief overview of the purpose and effect of the D Mode settings. For more detailed information about the D Mode settings, refer to D Mode Overview.

Serial Settings

- **Baud Rate**—Number of bits transmitted over the serial interface per second (300, 600, 1200, 2400, 4800, 9600, 9200, 38400, 57600, and 115200).

- **Data Bits**—Number of data bits that represent a serial byte.
  - **7**—Seven data bits per data byte.
  - **8**—Eight data bits per data byte.

- **Parity**—Parity algorithm used to detect serial errors.
  - **None**—No parity is used to detect serial errors.
  - **Even**—Data bits plus the parity bit sum to an even number.
  - **Odd**—Data bits plus the parity bit sum to an odd number.

- **Stop Bits**—Number of bits per byte used to indicate the end of the serial data byte.
  - **1**—One stop bit is used for every data byte.
  - **2**—Two stop bits are used for every data byte.

- **Flow Control**—Method used to ensure serial data is not lost.
  - **Hardware**—Hardware protocol used for flow control. For more information, refer to Hardware Handshaking.
  - **XonXoff**—Software implementation of the flow control protocol. For more information, refer to XON/XOFF Software Handshaking.
  - **None**—No flow control used.

The GPIB settings determine the GPIB interface behavior.

GPIB Settings

- **Listen Only**—GPIB talker and listener or GPIB listener only. All other GPIB settings have no effect if this setting is ON. For detailed information about this setting, refer to GPIB-RS232/485/422 Operation in Listen-Only Mode.
  - **ON**—The GPIB-RS232/485/422 acts as a listener and cannot be unaddressed to listen.
  - **OFF**—The GPIB-RS232/485/422 can be addressed to talk or listen.

- **GPIB Primary Address**—The GPIB-RS232/485/422 primary address.

- **GPIB Secondary Address**—The GPIB-RS232/485/422 secondary address.

- **SRQ**—Whether the GPIB-RS232/485/422 asserts SRQI under certain conditions. For detailed information about this setting, refer to Using the SRQ Feature.
  - **ON**—SRQI is asserted when the GPIB buffer is empty or the serial buffer is nonempty.
  - **OFF**—SRQI is never asserted.

Configuration of C Mode

When the GPIB-RS232/485/422 is configured for C Mode, it acts as a passthrough device that reads serial data and transfers it over GPIB. For more details about C Mode operation, refer to C Mode Overview.

C Mode requires the user to configure the serial settings used on the serial interface, as well as the GPIB settings used on the GPIB interface. Below is a brief overview of the C Mode settings. For more detailed information, refer to C Mode Overview.

The serial settings must exactly match the serial settings used on the host serial port. The serial settings are:

- **Baud Rate**—The number of bits transmitted over the serial interface per second (300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, or 115200).

- **Data Bits**—The number of data bits that represent a serial byte.
  - **7**—Seven data bits per data byte.
  - **8**—Eight data bits per data byte.

- **Parity**—The parity algorithm used to detect serial errors.
  - **None**—No parity is used to detect serial errors.
  - **Even**—Data bits plus the parity bit sum to an even number.
  - **Odd**—Data bits plus the parity bit sum to an odd number.

- **Stop Bits**—The number of bits per byte used to indicate the end of the serial data byte.
  - **1**—One stop bit is used for every data byte.
  - **2**—Two stop bits are used for every data byte.

- **Flow Control**—The method used to ensure serial data is not lost.
  - **Hardware**—The hardware protocol used for flow control. For more information, refer to Hardware Handshaking.
  - **XonXoff**—The software implementation of the flow control protocol. For more information, refer to XON/XOFF Software Handshaking.
  - **None**—No flow control used.

The GPIB settings determine the GPIB interface behavior. The GPIB settings are:

- **GPIB Primary Address**—The Primary Address of the device which the GPIB-RS232/485/422 will be communicating with.

- **GPIB Secondary Address**—The Secondary Address of the device which the GPIB-RS232/485/422 will be communicating with.

- **Termination Mode**—The termination algorithm that determines the GPIB-RS232/485/422 behavior. For detailed information about this setting, refer to Operating in C Mode.
  - **CR**—After receiving a `'\r'` over serial (0x0d) and sending it over GPIB, the GPIB-RS232/485/422 addresses the GPIB device as a talker so that it can send a response to the host.
  - **LF**—After receiving a `'\n'` over serial (0x0a) and sending it over GPIB, the GPIB-RS232/485/422 addresses the GPIB device as a talker so that it can send a response to the host.
  - **Timeout**—300 ms after the serial input buffer becomes empty, the GPIB-RS232/485/422 addresses the GPIB device as a talker so that it can send a response to the host.
  - **No swap**—Never address the device to talk.

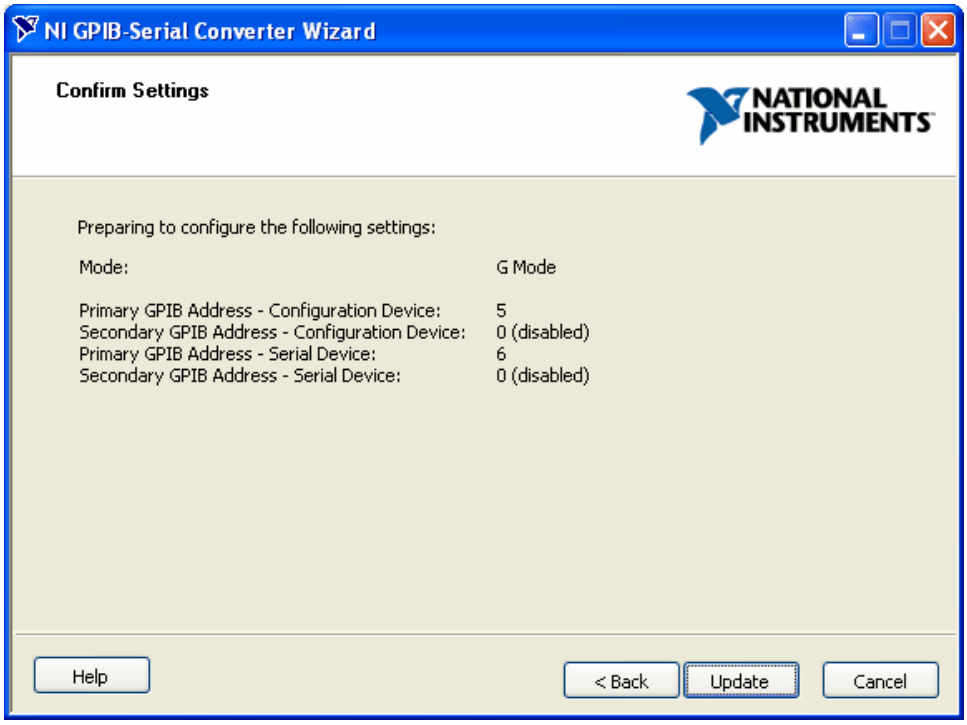
- **EOI**—Whether EOI is asserted on the last byte of a write to the device connected to the GPIB-RS232/485/422 (in affect for CR or LF termination modes only).

Verifying Settings

After completing the mode-specific setting configuration, a verification screen indicating your choices is displayed, as shown in the following figure.

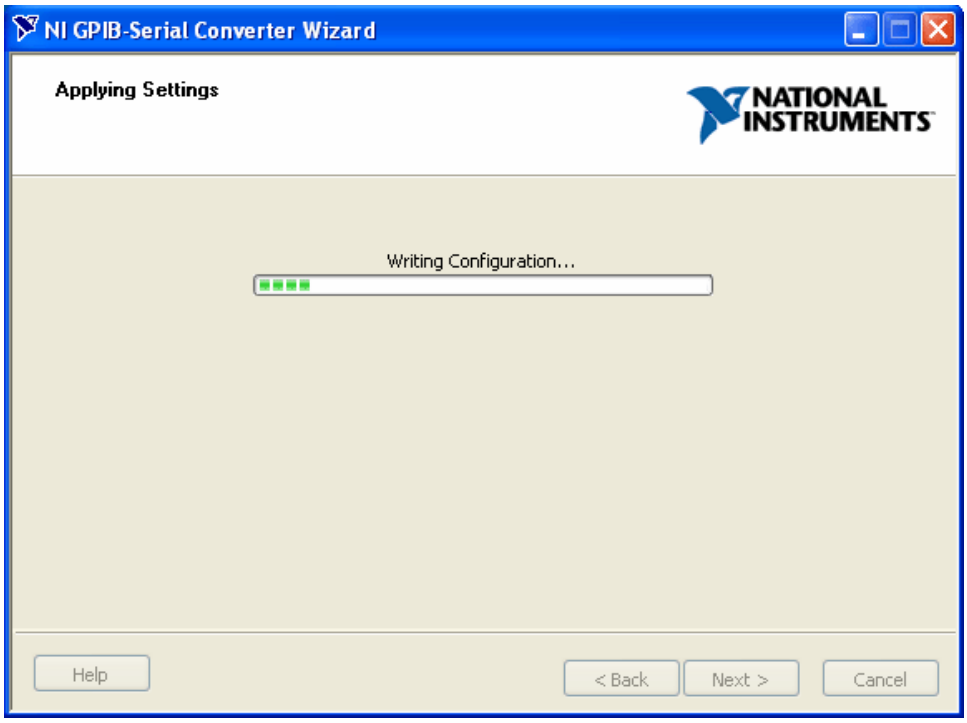
This screen allows you to review the choices prior to actually modifying the GPIB-RS232/485/422 configuration. Click **Update** and proceed to Applying Settings to configure the settings shown.

Note Depending on the mode and settings chosen, the screen displayed may not be identical to the following figure.

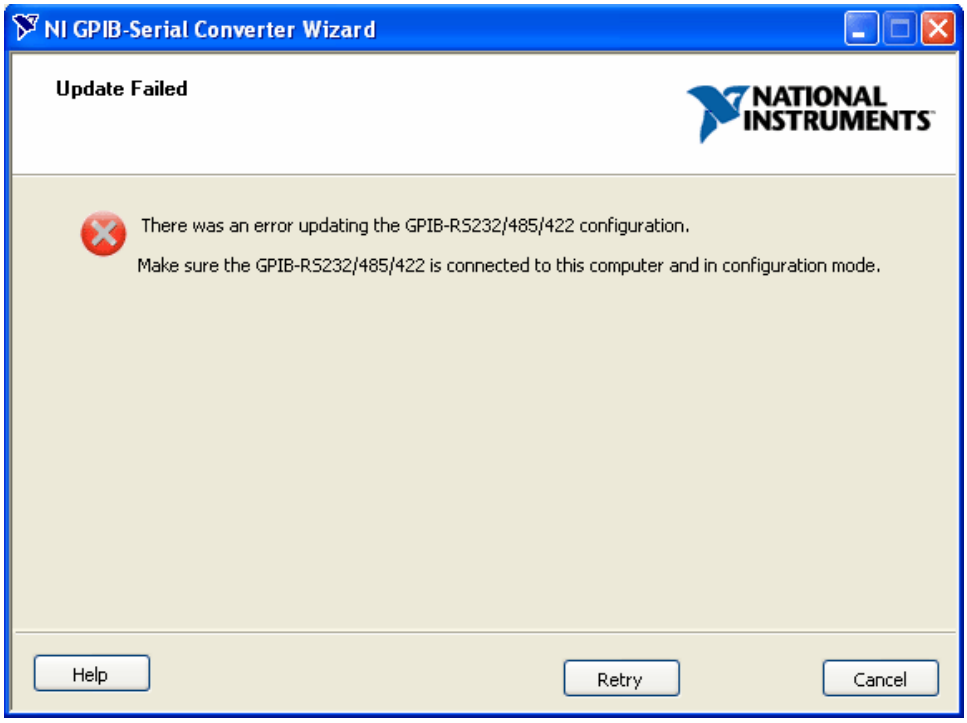


Applying Settings

In the following figure, the NI GPIB-Serial Converter Wizard is redetecting the GPIB-RS232/485/422 and issuing commands to update the GPIB-RS232/485/422 configuration. On successful completion, an **Update Successful** screen appears. Proceed to Using the GPIB-RS232/485/422 after Running the NI GPIB-Serial Converter Wizard.



If an error occurred during the configuration update, the There was an error updating the GPIB-RS232/485/422 configuration message displays, as shown in the following figure.



If the NI GPIB-Serial Converter Wizard appears as shown above, proceed to Update

Failed.

Update Failed

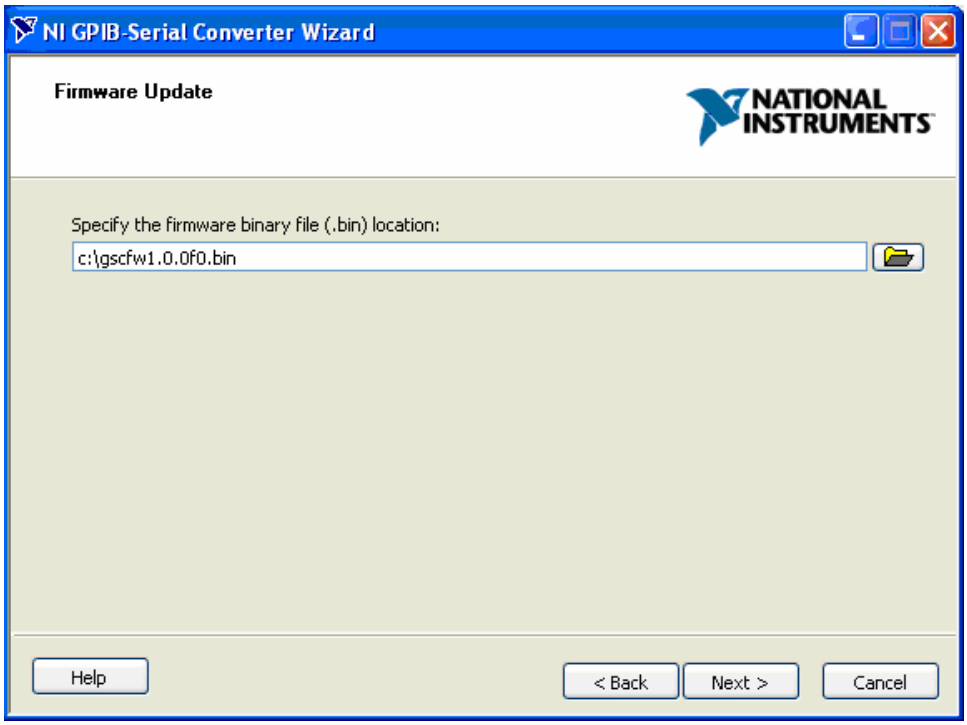
If an error occurred during the configuration update, the message There was an error updating the GPIB-RS232/485/422 configuration appears. Follow these steps to resolve the error and update the GPIB-RS232/485/422 configuration:

1. Verify that the GPIB-RS232/485/422 is connected to the computer interface specified in Selecting an Interface.
2. Verify that the GPIB-RS232/485/422 is in Configuration mode. The **RDY** LED is blinking green if the GPIB-RS232/485/422 is in Configuration mode. Refer to Connecting the GPIB-RS232/485/422 for instructions for enabling Configuration mode.
3. Verify that the cable connecting the computer to the GPIB-RS232/485/422 is connected correctly.
4. If connecting to the GPIB-RS232/485/422 over GPIB, verify that the GPIB interface used to configure the GPIB-RS232/485/422 is set to System Controller, is Controller In Charge of the GPIB bus, and has a Primary Address (PAD) other than 13.
5. If connecting to the GPIB-RS232/485/422 over GPIB, remove all devices from the GPIB bus other than your computer interface card and the GPIB-RS232/485/422.

After resolving the error causing the failure, click **Retry** to re-attempt the configuration update. If successful, proceed to Using the GPIB-RS232/485/422 After Running the NI GPIB-Serial Converter Wizard. If errors persist, recheck the troubleshooting steps above.

Using the NI GPIB-Serial Converter Wizard to Update the GPIB-RS232/485/422 Firmware

Before proceeding, the NI GPIB-Serial Converter Wizard must be running and your screen must appear as shown below. Refer to Launching the NI GPIB-Serial Converter Wizard for more details.



You also must have the latest firmware file. The firmware file is available from `www.ni.com/support`.

After you download the latest firmware file, select it by either entering the path or navigating to it using the file dialog. Click **Next** to validate the firmware image.

If the NI GPIB-Serial Converter Wizard successfully validates the specified firmware image file, it displays the **Confirm Firmware** screen, as shown in Confirm Firmware.

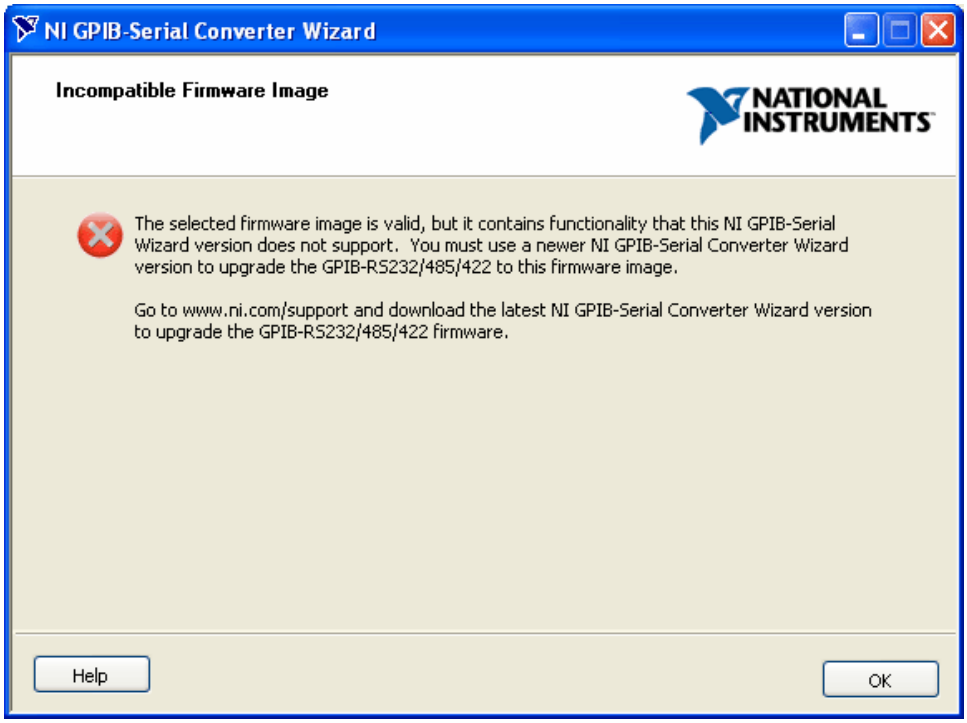
If the firmware image file chosen is a valid firmware file, but the new firmware version is not compatible with the NI GPIB-Serial Converter Wizard version you are running, the **Incompatible Firmware Image** screen is displayed, as shown in Incompatible Firmware Image.

If the firmware image file chosen is not a valid firmware file, the **File Error** screen is displayed, as shown in File Error.

Incompatible Firmware Image

If the firmware file chosen is a valid firmware file, but the new firmware version is not compatible with the NI GPIB-Serial Converter Wizard version you are running, the

Incompatible Firmware Image screen is displayed, as shown in the following figure.

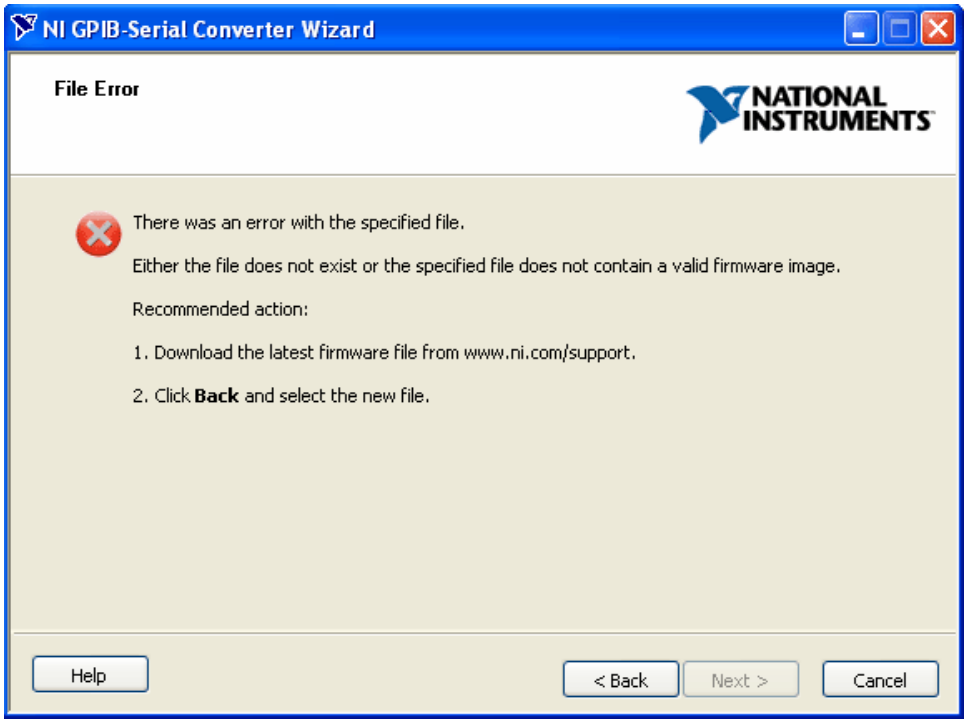


You must download a new version of the NI GPIB-Serial Converter Wizard to use the new firmware file. Follow these steps to download the new wizard version:

1. Exit the NI GPIB-Serial Converter Wizard.
2. Go to `www.ni.com/support`.
3. Locate and download the current NI GPIB-Serial Converter Wizard version.
4. Refer to Install the Software to install the new NI GPIB-Serial Converter Wizard version.
5. Refer to Launching the NI GPIB-Serial Converter Wizard to run the new NI GPIB-Serial Converter Wizard version.

File Error

If the firmware file chosen is not a valid firmware file, the **File Error** screen is displayed, as shown in the following figure.

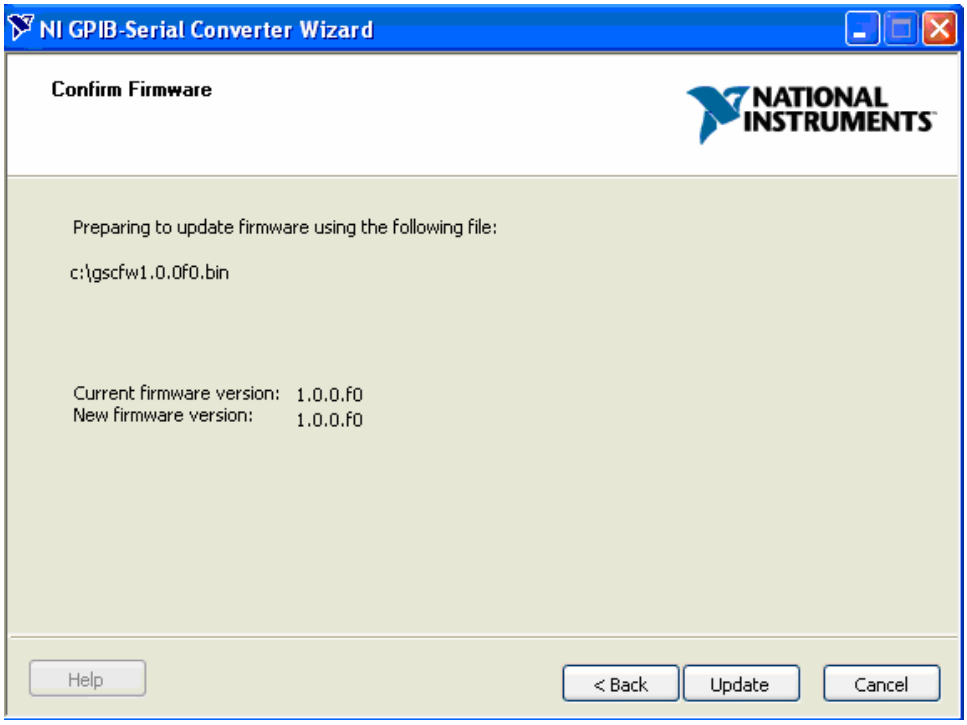


The most likely cause of this message is that you selected an incorrect file. Verify that you have selected the file downloaded from `www.ni.com/support`.

Another potential cause of this error is that the file has become corrupted during download. Re-download the firmware file, click **Back**, and select the new file.

Confirm Firmware

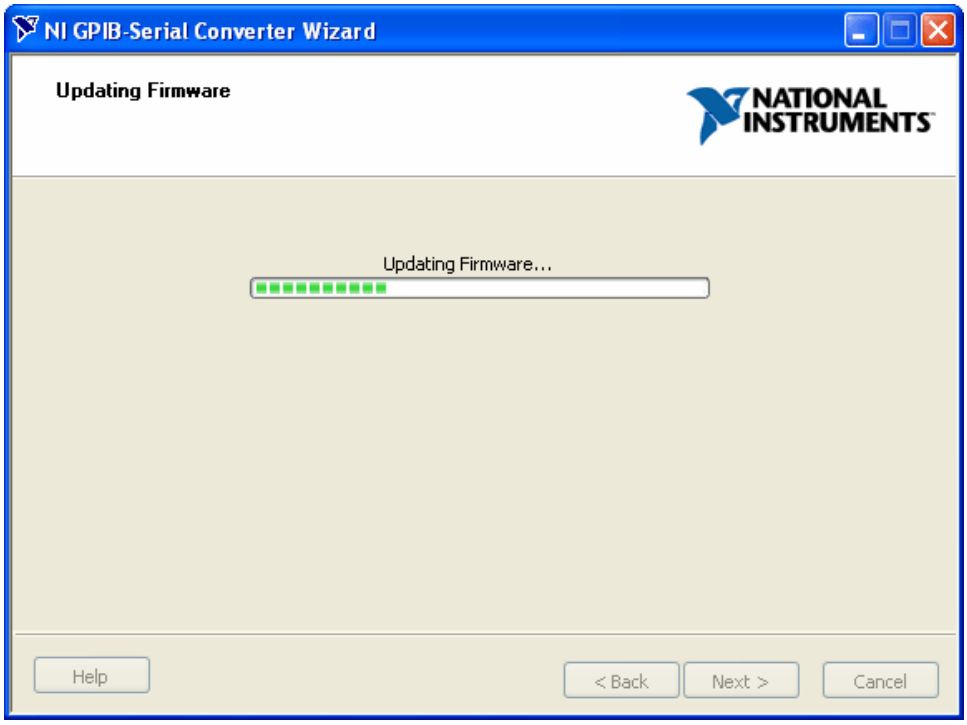
If the NI GPIB-Serial Converter Wizard successfully validates the specified firmware image file, it displays a **Confirm Firmware** screen, as shown in the following figure. This screen allows you to review the file choice and firmware versions prior to actually updating the GPIB-RS232/485/422 firmware. Click **Update** and proceed to Updating Firmware.



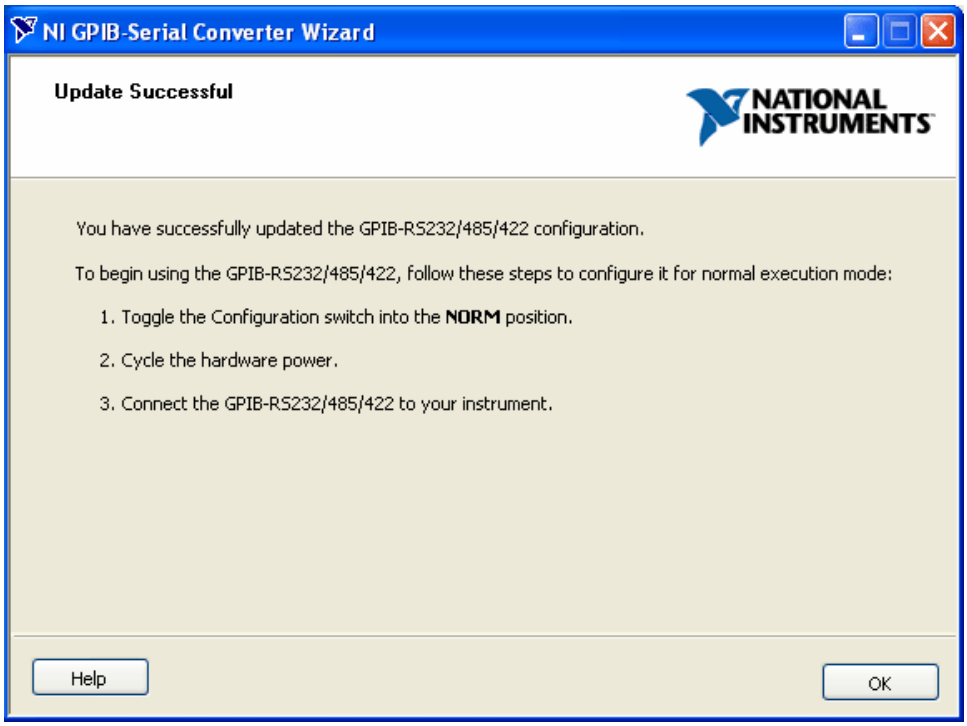
Note Depending on the firmware image selected and the current firmware version, the screen displayed will be similar to the figure above.

Updating Firmware

In the following figure, the NI GPIB-Serial Converter Wizard is redetecting the GPIB-RS232/485/422 and issuing commands to update the GPIB-RS232/485/422 firmware.

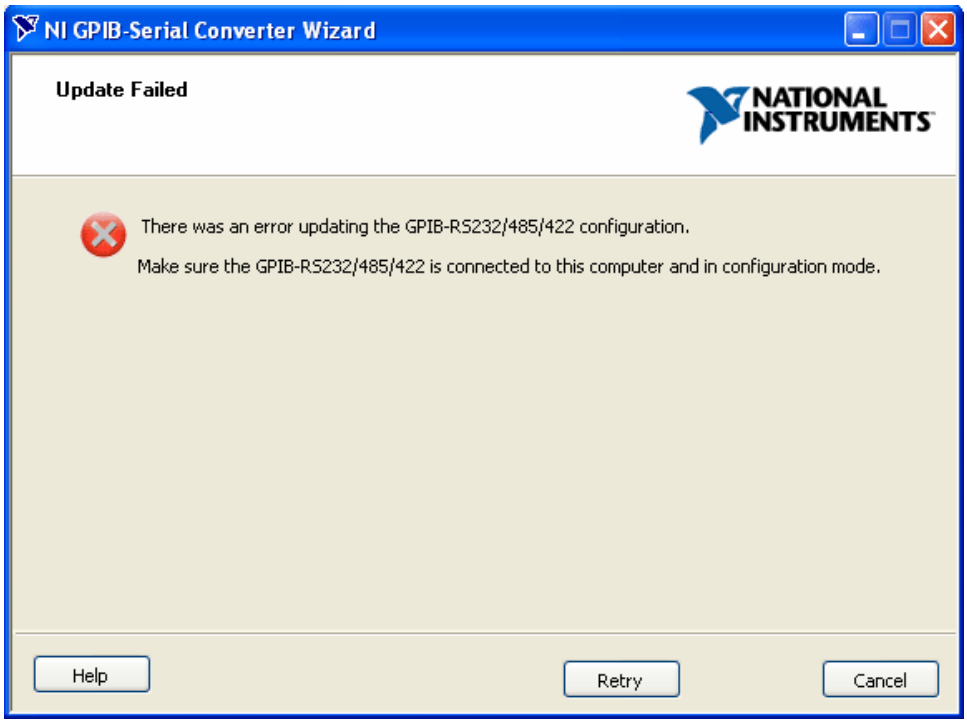


On successful completion, an **Update Successful** screen should appear, as shown in the following figure.



Proceed to Using the GPIB-RS232/485/422 after Running the NI GPIB-Serial Converter Wizard.

If an error occurred during the firmware update, the There was an error updating the GPIB-RS232/485/422 configuration message displays, as shown in the following figure.



If the NI GPIB-Serial Converter Wizard appears as shown above, proceed to Update Failed.

Update Failed

If an error occurred during the firmware update, the There was an error updating the GPIB-RS232/485/422 configuration message appears. Follow these steps to resolve the error and update the GPIB-RS232/485/422 firmware:

1. Verify that the GPIB-RS232/485/422 is connected to the computer interface specified in Selecting an Interface.
2. Verify that the GPIB-RS232/485/422 is in Configuration mode. The **RDY** LED is blinking green if the GPIB-RS232/485/422 is in Configuration mode. Refer to Connecting the GPIB-RS232/485/422 for instructions on enabling Configuration mode.
3. Verify that the cable connecting the computer to the GPIB-RS232/485/422 is connected correctly.
4. If connecting to the GPIB-RS232/485/422 over GPIB, verify that the GPIB interface being used to configure the GPIB-RS232/485/422 is set as System Controller, is

Controller In Charge of the GPIB bus, and has a Primary Address (PAD) other than 13.
5. If connecting to the GPIB-RS232/485/422 over GPIB, remove all devices from the GPIB bus other than your computer interface card and the GPIB-RS232/485/422.

After resolving the error causing the failure, click **Retry** to re-attempt the firmware update. If successful, proceed to Using the GPIB-RS232/485/422 After Running the NI GPIB-Serial Converter Wizard. If errors persist, recheck the troubleshooting steps above.

Using the GPIB-RS232/485/422 After Running the NI GPIB-Serial Converter Wizard

The NI GPIB-Serial Converter Wizard has now completed configuration. To use the GPIB-RS232/485/422, follow these steps:

1. Set the Configuration switch on the rear of the GPIB-RS232/485/422 to **NORM**. Refer to Configure the Hardware for the Configuration switch location.
2. Turn the Power switch to **OFF** to turn off the GPIB-RS232/485/422. Refer to LEDs for the Power switch location.
3. Turn the Power switch to **ON** to restart the GPIB-RS232/485/422.

You can now use the GPIB-RS232/485/422 in its current operating mode. For operating mode operation details, refer to the section corresponding to the current GPIB-RS232/485/422 mode.

Programming in S Mode

Programming in G Mode

D Mode Overview

C Mode Overview

Programming Background and Considerations for the GPIB-RS232/485/422

The following topics discuss programming steps and contain program examples you can use as guidelines as you start writing programs for the GPIB-RS232/485/422.

Programming Steps and Examples

General Programming Steps for Communicating with the GPIB-RS232/485/422 Over Serial

Serial Initialization Code Using VISA

General Programming Steps for Communicating with the GPIB-RS232/485/422 Over GPIB

GPIB Initialization Code Using VISA

Programming Steps and Examples

General Programming Steps for Communicating with the GPIB-RS232/485/422 over Serial applies to programmatically communicating with the GPIB-RS232/485/422 over serial. General Programming Steps for Communicating with the GPIB-RS232/485/422 over GPIB applies to communicating with the GPIB-RS232/485/422 over GPIB.

Programming examples in this help are in ANSI C using the VISA API for connectivity over GPIB and/or serial.

General Programming Steps for Communicating with the GPIB-RS232/485/422 Over Serial

The following topic includes general programming steps for creating a program that uses S Mode or C Mode operation.

Serial Initialization Code Using VISA

Serial Initialization Code Using VISA

```
#include "visa.h"
```

```
int main(void)
{

ViSession                           defaultRM;
ViSession                           instr;
ViUInt32                            retCount;
ViUInt32                            writeCount;
ViStatus                            status;
unsigned char                       buffer[100];
unsigned char                       stringinput[512];

/* Begin by initializing the system */
status = viOpenDefaultRM(&defaultRM);
if (status < VI_SUCCESS) {
/* Error Initializing VISA...exiting */
return -1;
}

/* Open communication with serial port at COM1.*/
status = viOpen(defaultRM, "ASRL1::INSTR", VI_NULL, VI_NULL, &instr);

/* Set Baud Rate and other serial port settings */
viSetAttribute(instr, VI_ATTR_ASRL_BAUD, 9600);
viSetAttribute(instr, VI_ATTR_ASRL_DATA_BITS, 8);
viSetAttribute(instr, VI_ATTR_ASRL_PARITY, VI_ASRL_PAR_NONE);
viSetAttribute(instr, VI_ATTR_ASRL_STOP_BITS, VI_ASRL_STOP_ONE);
viSetAttribute (instr, VI_ATTR_ASRL_FLOW_CNTRL, VI_ASRL_FLOW_RTS_CTS);
viSetAttribute (instr, VI_ATTR_TERMCHAR, 10);
viSetAttribute (instr, VI_ATTR_TERMCHAR_EN, VI_TRUE);
viSetAttribute (instr, VI_ATTR_ASRL_END_IN, VI_ASRL_END_TERMCHAR);

/*
TODO: Insert application specific code here
*/

/* Close down the system */
status = viClose(instr);
status = viClose(defaultRM);
return 0;
}
```

General Programming Steps for Communicating with the GPIB-RS232/485/422 Over GPIB

The following topic includes general programming steps for creating a program that uses G Mode or D Mode operation.

GPIB Initialization Code Using VISA

GPIB Initialization Code Using VISA

```
#include "visa.h"

int main(void)
{
ViStatus   status;
ViSession  defaultRM;
ViSession  GPIB232;
ViUInt32   retCount;

/* Open Default RM */
status = viOpenDefaultRM(&defaultRM);
if (status < VI_SUCCESS) {

/* Error Initializing VISA...exiting */
return -1;
}

/* Open up handle to GPIB-RS232/485/422 at GPIB Primary Address 1 */
status = viOpen(defaultRM, "GPIB::1::INSTR", VI_NULL, VI_NULL, &GPIB232);

/* TODO: Add application specific code here */

/* Use device and eventually close it. */
viClose(GPIB232);
viClose(defaultRM);
return 0;
}
```

Programming in S Mode

The following topics describe how to program the GPIB-RS232/485/422 in S Mode. They explain status information and error handling, programming considerations, programming messages, function arguments, GPIB termination methods, function names, S Mode default settings, and the S Mode functions.

Status Information and Error Handling Characteristics

Programming Considerations

Programming Messages

Programming Message Format

Programming Message Example

Programming Message Example with Data String

How Messages are Processed

Function Arguments

Abbreviations for Arguments

GPIB Address

Lists of GPIB Addresses

Numeric String Arguments

GPIB Read and Write Termination Methods (END and EOS)

Function Names

S Mode Default Settings and Related Functions

List of S Mode Functions by Group

GPIB Functions

Serial Port Functions

General-Use Functions

Alphabetical List of S Mode Functions

Status Information and Error Handling Characteristics

The function descriptions in S Mode Functions explain that the GPIB-RS232/485/422 records specific status and error information. This means it stores the information in its memory so the information is available when you request it.

The function descriptions also explain that the GPIB-RS232/485/422 returns certain information to you. This means the GPIB-RS232/485/422 sends information to you over the serial port.

The GPIB-RS232/485/422 continuously monitors the serial port for transmission errors. If it encounters an error in the serial data, the GPIB-RS232/485/422 records the error. You can program the GPIB-RS232/485/422 to ignore serial port errors using the `spign` function.

Programming Considerations

The programming examples for each function description are in standard ANSI C, using NI-VISA. Although the examples in this help are written in C, you can program the GPIB-RS232/485/422 using any programming language with access to a serial port.

In the function syntax descriptions, arguments enclosed in square brackets (`[]`) are optional. Do not enter the brackets as part of your argument.

For all programming examples, the communications port has been opened using initialization code found in General Programming Steps for Communicating with the GPIB-RS232/485/422 Over Serial.

The I/O and high-level bus management functions are the most frequently used and should meet most of your needs. In the function descriptions, these functions are marked with an asterisk (*).

You can use function name abbreviations, which include only as many characters as necessary to distinguish them from other functions. The abbreviated forms are indicated by bold text in the syntax description of each function.

Programming Messages

You can program the GPIB-RS232/485/422 by sending programming messages (ASCII strings) and data strings to its serial port.

Programming Message Format

Each programming message is terminated with a carriage return (<CR>), a linefeed (<LF>), or a carriage return followed by a linefeed (<CR><LF>).

Message termination is indicated by a <CR> in the syntax portions of the function descriptions.

You can enter programming messages in any combination of uppercase and lowercase letters.

Programming Message Example

The following line of code is an example programming message in C using NI-VISA:

```
status = viWrite(instr, "clr 3,4\r", 8, &retCount);
```

This programming message contains the function name `clr` and the arguments `3` and `4`. It tells the GPIB-RS232/485/422 to clear the devices at GPIB addresses 3 and 4. `viWrite` is the NI-VISA function to send characters to the serial port after opening the serial port with the serial initialization code.

Programming Message Example with Data String

The following lines of code are an example of a programming message with a data string:

```
viWrite(instr, "wrt 2\n",6, &retCount);
viWrite(instr, "*idn?\r",6, &retCount);
```

This programming message contains the function name `wrt`, the argument `2`, and the data string `"*idn?"`. It tells S Mode to write to the device at primary address 2. `"*id`

n?" is the data string that contains the data wrt sends out on the GPIB. In this case, a carriage return "\r" is added to the data string to terminate the wrt command.

Both the cmd and wrt programming messages are followed by a data string that can contain 7 or 8-bit data.

How Messages Are Processed

The GPIB-RS232/485/422 processes a programming message on a line-by-line basis. When the GPIB-RS232/485/422 receives a message, it buffers the entire message, interprets the function name and arguments, and then executes the message. The data portions of the wrt and cmd functions are processed differently. The data immediately following a wrt and a cmd function are sent directly to the GPIB.

S Mode recognizes <CTRL-H> (hex 8) in a programming message as a backspace and erases the previous character. S Mode recognizes <CTRL-H> in a data string as a data byte and does not erase the previous character.

Function Arguments

When specifying a function, separate the first argument from the function name with at least one space. Separate each additional argument with at least one space or a comma.

In the syntax portions of the function descriptions in S Mode Functions, the information within the square brackets ([]) is optional. When including optional information, do not include the brackets as part of your argument.

Abbreviations for Arguments

The function descriptions in S Mode Functions use abbreviations for some arguments. The abbreviations are as follows:

addr	A GPIB address
alist	one or more addresses (addrs)

`bool`	a Boolean value (1 = true, on, or enable; 0 = false, off, or disable)

GPIB Address

One argument used with most functions is the GPIB address. Each device on the GPIB has a GPIB address. The GPIB-RS232/485/422 address is 0 at power on, but you can change it using the `caddr` function. Refer to the GPIB device manuals for the device addresses. You should know the addresses when you program the GPIB-RS232/485/422.

Only the lower five bits of each GPIB address are significant. These bits can range from 0 through 30 for both the primary and secondary address.

For example, the binary value 01100010 (decimal 98) is interpreted as decimal 2.

Each of the following GPIB addresses specifies a primary address of 0 and a secondary address of 2. A plus sign (+) separates the primary address from the secondary address:

- 0+2
- 0+98
- 32+98
- 0+\x62

Lists of GPIB Addresses

When a function requires a list of one or more GPIB addresses, the maximum number of addresses you can specify is 14. If you specify more than 14 addresses, the GPIB-RS232/485/422 records the EARG error. This limit exists because the IEEE 488 specification allows no more than 15 GPIB devices to coexist on any GPIB.

Numeric String Arguments

Another type of argument is a numeric string. A numeric string represents an integer you can express using decimal, octal, or hexadecimal digits. To specify an octal integer, precede the numeric string with a backslash (`\`). To specify a hexadecimal integer, precede the numeric string with a backslash X (`\X`).

Each of the following numeric strings represents the decimal value 112:

- Decimal `112`
- Octal `\160`
- Hex `\x70`

GPIB Read and Write Termination Methods (END and EOS)

The IEEE 488 specification defines two ways that GPIB Talkers and Listeners can identify the last byte of data messages: END and EOS. The two methods permit a Talker to send data messages of any length without the Listener(s) knowing the number of transmission bytes in advance. You can use END and EOS individually or in combination, but you must configure the Listener to detect the end of a transmission.

END message	The Talker asserts the EOI* (End Or Identify) signal while the last data byte is being transmitted. The Listener stops reading when it detects a data decimal octal hexadecimal byte accompanied by EOI*, regardless of the byte value.
EOS character	The Talker sends an EOS (end-of-string) character at the end of its data string. The Listener stops receiving data when it detects the EOS character. You can use either a 7-bit ASCII character or a full 8-bit binary byte.

The GPIB-RS232/485/422 always terminates GPIB `rd` operations on the END message. You can use the `eos` and `eot` functions to change GPIB read and write termination methods.

Function Names

Each function name has been selected to indicate the function's purpose and make your programs easier to understand. However, to reduce some overhead in your program, you can use an abbreviation of the name that includes only as much of the function name as is necessary to distinguish it from other functions. This abbreviated form of the function name is shown in bold text in the function tables and syntax portions of the function descriptions.

S Mode Default Settings and Related Functions

The following tables list power-on characteristics of the GPIB-RS232/485/422 in S Mode and the functions you can use to change those characteristics.

S Mode Serial Port Characteristics

Characteristic	Default Setting	Related Function
Echo bytes to serial port	No	`echo`
Ignore serial port errors	Yes	`spign`
Send XON/XOFF	Configurable*	`xon`
Recognize XON/XOFF	Configurable*	`xon`

* You can configure the power-on value of this setting using the NI GPIB-Serial Converter Wizard. For more information, refer to Configuration of S Mode, Hardware Handshaking, and Serial Flow Control in S Mode.

S Mode GPIB Characteristics

Characteristic	Default Setting	Related Function
Primary/secondary address	pad = 0, sad = none	`caddr`
IEEE 488 parallel poll subset	PP1 (remote)	`conf`
End-of-string modes	None	`eos`
Send END on writes	Yes	`eot`
IST bit setting	0	`ist`
GPIB-RS232/485/422 is System Controller	Yes	`rsc`
I/O timeout	10 s	`tmo`
Serial poll timeout	0.1 s	`tmo`

List of S Mode Functions by Group

The GPIB-RS232/485/422 S Mode functions are divided into three main groups: GPIB functions, serial port functions, and general-use functions. For more information about the S Mode functions, refer to Alphabetical List of S Mode Functions or S Mode Functions.

GPIB Functions

Serial Port Functions

General-Use Functions

GPIB Functions

The GPIB functions manage the GPIB-RS232/485/422 GPIB port. The GPIB function subgroups are listed with the most frequently used groups first. Often, the I/O and high-level bus management functions are the only ones you need.

I/O Functions—Read data from and write to GPIB devices.

- rd
- wrt

High-Level Bus Management Functions—Send frequently used bus management instructions to GPIB devices.

- clr
- loc
- trg

GPIB Initialization Functions—Initialize various configurations of the GPIB port.

- caddr
- eos
- eot
- onl
- rsc
- tmo

Serial Poll Functions—Conduct and respond to GPIB serial polls.

- rsp
- rsv

Low-Level Bus Management Functions—Give you precise control over the GPIB. Use them when the I/O and high-level bus management functions do not meet the needs of your GPIB device.

- cac
- cmd
- gts
- lines
- ln
- pct
- sic
- sre

Parallel Poll Functions—Conduct and respond to GPIB parallel polls.

- ist
- ppc
- ppu
- rpp

Serial Port Functions

The serial port functions initialize and manage the GPIB-RS232/485/422 serial port.

- echo
- spign
- xon

General-Use Functions

The general-use functions are for general operations that the GPIB functions or serial port functions do not provide.

- conf
- id
- stat
- wait

Alphabetical List of S Mode Functions

The following table lists the S Mode functions in alphabetical order.

Function	Purpose
cac mode	Become Active Controller
caddr address	Change the GPIB address of the GPIB-RS232/485/422
clr address list	Clear specified device(s)
cmd count commands	Send GPIB commands
conf option value	Read/change configuration
echo on/off	Echo characters received from serial port
eos modes, eoschar	Change or disable GPIB end-of-string termination mode
eot on/off	Enable or disable END termination message on GPIB write operations
gts mode	Go from Active Controller to Standby
id	Identify system
ist set/clear	Set or clear individual status bit for use in GPIB-RS232/485/422 response to parallel polls
lines	Determine state of GPIB control lines
ln address list	Check for listening devices
loc address list	Go to Local
onl on/off	Place the GPIB-RS232/485/422 online/offline
pct address	Pass Control
ppc values	Parallel Poll Configure
ppu address list	Parallel Poll Unconfigure
rd count, address	Read data
rpp	Conduct (request) a Parallel Poll
rsc on/off	Request System Control
rsp address list	Conduct (request) a serial poll
rsv status byte	Request service and/or set or change the serial poll status byte

sic time	Send Interface Clear
spign on/off	Ignore serial port errors
sre on/off	Set remote enable
stat modes	Return GPIB-RS232/485/422 status
tmo values	Change or disable time limits
trg address list	Trigger selected device(s)
wait mask	Wait for selected event(s)
wrt count, address list, data	Write data
xon modes	Change serial port XON/XOFF protocol

S Mode Functions

The S Mode function descriptions describe the S Mode functions you can use to program the GPIB-RS232/485/422.

The I/O and high-level bus management functions are the most frequently used and should meet most of your needs.

For general information about using S Mode functions, refer to Programming in S Mode.

cac

Become Active Controller

Type

Low-level bus management function

Syntax

cac [bool]<CR>

Purpose

You can use `cac` to change the GPIB-RS232/485/422 from Standby Controller to Active Controller when the I/O and high-level bus management functions do not meet the needs of your device. `cac` gives you more precise control over the GPIB than the I/O and high-level bus management functions.

Remarks

If the argument `bool` is `0`, the GPIB-RS232/485/422 takes control immediately—that is, it takes control asynchronously. If the argument `bool` is `1`, the GPIB-RS232/485/422 takes control after any handshake that is in progress completes—that is, it takes control synchronously.

If you call `cac` without an argument, the GPIB-RS232/485/422 returns the current Controller status. The Controller status is 0 if the GPIB-RS232/485/422 is not the Active Controller and 1 if the GPIB-RS232/485/422 is the Active Controller. The response is terminated with carriage return and line feed.

The GPIB-RS232/485/422 must be CIC when you call `cac` with an argument. If you call `cac` with an argument and the GPIB-RS232/485/422 is not CIC, the GPIB-RS232/485/422 records the ECIC error.

The power-on Controller status of the GPIB-RS232/485/422 is Idle Controller.

See Also

`gts` and `sic`

Example 1

```
/* take control immediately */
char  buffer [] = "cac 0\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

Example 2

```
/* take control synchronously */
char  buffer [] = "cac 1\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

Example 3

```
/* Is S Mode the Active Controller? */
char  buffer [] = "cac\r";
char cacString[4];
viWrite(instr, buffer, strlen(buffer), &retCount);
viRead(instr, cacString, 4, &retCount);
if(cacString[0] == '0')
{
   printf("S Mode is not active controller\n");
}
else if (cacString[0] == '1')
{
   printf("S Mode is active controller\n");
}
```

caddr

Change the GPIB Address of GPIB-RS232/485/422

Type

Initialization function

Syntax

cad`dr [addr]`<CR>

Purpose

You can use `caddr` at the beginning of your program to change the GPIB-RS232/485/422 GPIB address.

Remarks

The argument `addr` is a device address that specifies the new GPIB address for the GPIB-RS232/485/422. `addr` consists of a primary address and an optional secondary address. The secondary address is separated from the primary address by a plus sign (+). Both addresses are expressed as numeric strings.

If you specify a primary address without a secondary address, secondary addressing is disabled.

If you call `caddr` without an argument, the GPIB-RS232/485/422 returns its current GPIB address.

The address assigned by this function remains in effect until you call `caddr` again, call `onl`, or turn off the GPIB-RS232/485/422.

The power-on default is 0 with secondary addressing disabled.

Example 1

Give the GPIB-RS232/485/422 a primary address of 0 and a secondary address of 22.

```
char  buffer [] = "caddr 0+22\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

Example 2

Change the GPIB-RS232/485/422 primary address to 1 and disable secondary addressing.

```
char  buffer [] = "caddr 1\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

Example 3

Get the GPIB-RS232/485/422 GPIB address.

```
char  buffer [] = "caddr\r";
char caddrString[10];
memset(caddrString,0,10);
viWrite(instr, buffer, strlen(buffer), &retCount);
viRead(instr, caddrString, 10, &retCount);
printf("S Mode address is %s", caddrString);
```

clr

Clear Specified Device

Type

High-level bus management function

Syntax

cl`r [alist]`<CR>

Purpose

You can use `clr` to reset the internal or device functions of the specified GPIB devices. For example, a multimeter might require that you send it either the GPIB Device Clear or Selected Device Clear command to change its function, range, and trigger mode back to its default setting.

Remarks

The argument `alist` is a list of addresses separated by commas or spaces. addresses specify the GPIB addresses you want to clear.

If you call `clr` with `alist`, the GPIB-RS232/485/422 clears only the devices specified in `alist` (Selected Device Clear). If you call `clr` without `alist`, the GPIB-RS232/485/422 clears all devices (Device Clear).

If `clr` is the first function you call that requires GPIB Controller capability, and you have not disabled System Controller capability with `rsc`, the GPIB-RS232/485/422

sends Interface Clear (`IFC*`) to make itself CIC. It also asserts Remote Enable.

If you passed control to some other GPIB device, control must be passed back to you or you must send `IFC*` to make yourself CIC before making this call. Otherwise, the ECIC error is recorded.

See Also

GPIB Basics for more information about clearing devices, and Status and Error Message Information for more error information.

Example 1

Selectively clear five devices.

```
char  buffer [] = "clr 14+30,16+12,18,3+26,6\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

Example 2

Issue a device clear to all devices.

```
char  buffer [] = "clr\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

cmd

Send GPIB Commands

Type

Low-level bus management function

Syntax

cm`d [#count]`<LF>
`commands`<CR>

Purpose

You can use cmd when the I/O and high-level bus management functions do not meet the needs of your device. cmd gives you precise control over the GPIB. For example, in applications that require command sequences not sent by other functions, you can use cmd to transmit any sequence of interface messages (commands) over the GPIB.

Remarks

The argument commands is a list of GPIB commands. These commands are represented by their ASCII character equivalents. For example, the GPIB Untalk (UNT) command is the ASCII character underscore (_). The GPIB commands, or interface messages, are listed in Multiline Interface Messages. They include device talk and listen addresses, secondary addresses, device clear and trigger messages, and other management messages.

The argument count is a numeric string preceded by a number sign (#). count specifies the number of GPIB command bytes (interface messages) to send, and can range from 1 to 4294967295. count must not include the <LF> or <CR> characters, which are used to terminate the programming message or string of GPIB command bytes.

If you call cmd without count, the GPIB-RS232/485/422 recognizes the end of the command string when it sees a <CR> or an <LF>. Consequently, count is required only if the command string contains a <CR> or <LF> character.

Do not terminate the cmd programming message with the <CR> character alone. Use either <LF> or <CR><LF> to terminate the programming message. If you use <CR> alone and the first character of the command string is <LF>, the GPIB-RS232/485/422 discards the <LF> as the second character of a <CR><LF> termination.

Do not use cmd to send programming instructions to devices. Use rd and wrt to send or receive device programming instructions and other device-dependent information. If the GPIB-RS232/485/422 is CIC but not Active Controller, it takes control and asserts ATN* before sending the command bytes. It remains Active Controller afterward.

The cmd operation terminates when:

- The GPIB-RS232/485/422 successfully transfers all commands.
- The GPIB-RS232/485/422 is not Controller-In-Charge. The ECIC error is recorded.
- The I/O time limit is exceeded. The EABO error is recorded.
- There is no device on the bus receiving the command bytes. The ENOL error is recorded.
- A serial port error occurs and is not ignored (refer to `spign`).

After `cmd` terminates, the GPIB-RS232/485/422 records the number of command bytes it actually sent. If one of the errors described above occurs, the count might be less than expected.

If an error occurs and the GPIB-RS232/485/422 cannot transmit the entire command string, the GPIB-RS232/485/422 reads in and discards the remaining bytes of the command string.

See Also

`spign` and Multiline Interface Messages.

Example 1

Program the device at address 11 to listen and GPIB-RS232/485/422 at address 0 to talk. Then write the string `ABCDE` to the device at address 11.

```
char  cmdBuffer [] = "cmd\n+@\r";
char  wrtBuffer [] = "wrt\nABCDE\r";
viWrite(instr, cmdBuffer, strlen(cmdBuffer), &retCount);
viWrite(instr, wrtBuffer, strlen(wrtBuffer), &retCount);
```

Example 2

Pass control to the device at GPIB address 23. 0x9 is the TCT command.

```
char  buffer [] = "cmd #4\n_?W\x9\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

conf

Read/Change GPIB-RS232/485/422 Configuration

Type

General-use function

Syntax

co`nf option [value]`<CR>

Purpose

You can use `conf` to read or change certain configuration parameters of the GPIB-RS232/485/422.

Remarks

The argument `option` indicates which configuration parameter to read or change. The argument `value` indicates the new setting of the configuration parameter indicated by `option`. The `value` argument is optional. If you do not specify `value`, the current setting for the specified `option` is returned.

When `option` is `0`, the PP2 option is indicated. The PP2 option indicates what type of parallel poll configurations the GPIB-RS232/485/422 should accept. If the PP2 `value` is `0`, the GPIB circuitry of the GPIB-RS232/485/422 uses the IEEE 488 Parallel Poll (PP) interface function subset PP1 (remote configuration from an external Controller). If the PP2 `value` is `1`, the GPIB-RS232/485/422 uses PP subset PP2 (local configuration via the `ppc` and `ppu` functions). When PP subset PP2 is used, the GPIB-RS232/485/422 ignores remote parallel poll configurations. The default value of the PP2 option is `0`. The values assigned by this function remain in effect until you call `conf` again, call `on 1`, or turn off the GPIB-RS232/485/422.

See Also

`ppc`, `ppu`, and Parallel Polling.

The following examples show commands as you would enter them at a terminal.

Example 1

Enable the PP2 option.

```
char  buffer [] = "conf 0 1\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

Example 2

Return the current setting for the PP2 option.

```
char  buffer [] = "conf 0\r";
char pp2ModeString[5];
memset(pp2ModeString,0,5);
viWrite(instr, buffer, strlen(buffer), &retCount);
viRead(instr, pp2ModeString, 4, &retCount);
printf("The current setting for PP2 is %s", pp2ModeString);
```

echo

Echo Characters Received from Serial Port

Type

Serial port function

Syntax

ec`ho [bool]`<CR>

Purpose

You can use `echo` when a terminal is connected to the GPIB-RS232/485/422 and you want everything you type to display on the terminal screen.

Remarks

If the argument `bool` is `1`, characters received from the serial port are echoed back to the serial port. If the argument `bool` is `0`, characters are not echoed. If the argument `bool` is `1` and echoing was previously disabled, characters are not echoed until this command has been completely processed—that is, the next programming message is echoed.



Note There are minor behavioral differences between the GPIB-RS232/485/422 and GPIB-232/485CT/CV-A regarding this function. For more information about this and other behavioral differences, refer to Differences Between the GPIB-232/485CT/CV-A and GPIB-RS232/485/422.

If you call `echo` without an argument, the GPIB-RS232/485/422 returns the current setting.

The assignment made by `echo` remains in effect until you call `echo` again, call `onl`, or turn off the GPIB-RS232/485/422.

The following examples show commands as you would enter them at a terminal.

Example 1

```
echo 1<CR> 'Turn on character echoing.
```

Example 2

```
echo 0<CR> 'Disable character echoing.
```

Example 3

```
echo<CR> 'What is the current echo status?
response: 0<CR><LF> (character echo is disabled)
```

eos

Change/Disable GPIB EOS Termination Mode

Type

Initialization function

Syntax

eos `[R[B] eoschar]`<CR>

or

eos `X[B] eoschar`<CR>

or

eos `D`<CR>

Purpose

You can use `eos` at the beginning of your program if you want to use an `eos` mode when you transfer data to and from the GPIB. `eos` tells the GPIB-RS232/485/422 when to stop reading information from the GPIB. `eos` also enables the GPIB-RS232/485/422 to tell other devices that it is finished writing information to the GPIB. `eos` defines a specific character, end-of-string (EOS), to be recognized as a string terminator.

Remarks

The arguments `R`, `X`, `B`, and `D` specify GPIB termination methods. They enable or disable the corresponding `eos` mode. If a particular letter is specified, the corresponding `eos` mode is enabled. If it is not specified, the corresponding `eos` mode is disabled. By default, all `eos` modes are disabled. The following table lists the termination methods and corresponding letters.

Description	Letter
`REOS`—terminate read when EOS is detected	`R`
`XEOS`—set EOI* with EOS on write functions	`X`
`BIN`—compare all 8 bits of EOS byte rather than	`B`

low 7 bits (all read and write functions)	
DISABLE—disable all eos modes	D

Methods R and B determine how GPIB read operations (rd) performed by the GPIB-RS232/485/422 terminate. If method R alone is chosen, reads terminate when the low 7 bits of the byte that is read match the low 7 bits of the EOS character. If both methods R and B are chosen, a full 8-bit comparison is used.

Methods X and B together determine when GPIB write operations (wrt) performed by the GPIB-RS232/485/422 send the END message. If method X alone is chosen, the END message is sent automatically with the EOS byte when the low 7 bits of that byte match the low 7 bits of the EOS character. If methods X and B are chosen, a full 8-bit comparison is used.

eoschar is a numeric string that represents a single ASCII character. For example, decimal 10 represents the ASCII linefeed character. Refer to Multiline Interface Messages for more ASCII codes.

If you call eos without an argument, the GPIB-RS232/485/422 returns the current eos settings.

If you call eos with B alone as an argument, the GPIB-RS232/485/422 records the EARG error.

Note Defining an EOS byte for the GPIB-RS232/485/422 does not cause the GPIB-RS232/485/422 to insert that byte into the data string when performing GPIB writes. To send the EOS byte, you must include it in the data string that you send following the wrt programming message.

The assignment made by this function remains in effect until you call eos again, call onl, or turn off the GPIB-RS232/485/422.

See Also

GPIB Read and Write Termination Methods (END and EOS).

Example 1

Configure the GPIB-RS232/485/422 to terminate read when <LF> is detected, compare all 8 bits, and not send EOI.

```
char  buffer [] = "eos R,B,10\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

Example 2

Configure the GPIB-RS232/485/422 to assert EOI* with <CR> on `wrt`, doing a comparison on 7 bits. There is no EOS termination on reads.

```
char  buffer [] = "eos X,13\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

Example 3

Get the current EOS settings.

```
char  buffer [] = "eos\r";
char  eosSettings[10];
memset(eosSettings,0,10);
viWrite(instr, buffer, strlen(buffer), &retCount);
viRead(instr, eosSettings, 10, &retCount);
printf("EOS settings are: %s\n", eosSettings);
```

eot

Enable/Disable END Message on GPIB Writes

Type

Initialization function

Syntax

eot `[bool]`<CR>

Purpose

You can use `eot` at the beginning of your program to change how the GPIB-RS232/485/422 terminates GPIB writes. Using `eot`, you tell the GPIB-RS232/485/422 to automatically send or not send the GPIB END message with the last byte that it writes to the GPIB.

Remarks

If the argument `bool` is `1`, the GPIB-RS232/485/422 automatically sends the END message with the last byte of each `wrt`. If the argument `bool` is `0`, END is not sent. The power-on default is `1`.

The GPIB-RS232/485/422 sends the END message by asserting the GPIB EOI* signal during the last byte of a data transfer. `eot` is used primarily to send variable length data.

If you call `eot` without an argument, the GPIB-RS232/485/422 returns a `1` to indicate END termination is currently enabled, or a `0` to indicate END termination is currently disabled.

The assignment made by `eot` remains in effect until you call `eot` again, call `onl`, or turn off the GPIB-RS232/485/422.

See Also

GPIB Read and Write Termination Methods (END and EOS).

Example 1

Disable END termination.

```
char  buffer [] = "eot 0\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

Example 2

Assert EOI* with the last byte sent.

```
char  buffer [] = "eot 1\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

Example 3

Get the current EOT setting.

```
char  buffer [] = "eot\r";
char  eotSetting[5];
memset(eotSetting,0,5);
viWrite(instr, buffer, strlen(buffer), &retCount);
viRead(instr, eotSetting, 5, &retCount);
printf("EOT setting is: %s\n", eotSetting);
```

gts

Go from Active Controller to Standby

Type

Low-level bus management function

Syntax

g`ts [bool]`<CR>

Purpose

You can use `gts` to change the GPIB-RS232/485/422 from Active Controller to Standby

Controller. Use `gts` when the I/O and high-level bus management functions do not meet the needs of your device.

`gts` permits GPIB devices to transfer data without the GPIB-RS232/485/422 participating in the transfer. For example, you can use `gts` if you want to let two external devices talk to each other directly. The GPIB-RS232/485/422 can selectively participate in the handshake of the data transfer and hold off the handshake when it detects the END message. The GPIB-RS232/485/422 can then take control synchronously without possibly corrupting the transfer.

Remarks

If the argument `bool` is `1`, shadow handshaking is enabled. If the argument `bool` is `0`, shadow handshaking is not performed.

`gts` causes the GPIB-RS232/485/422 to go to the Controller Standby state and to unassert the ATN* signal if it is initially the Active Controller.

If you enable shadow handshaking, the GPIB-RS232/485/422 participates in the data handshake as an Acceptor without actually reading the data. It monitors the transfers for the END (EOI* or end-of-string character) message and holds off subsequent transfers. This mechanism allows the GPIB-RS232/485/422 to take control synchronously on a subsequent operation such as `cmd` or `rpp`.

Before performing a `gts` with a shadow handshake, you should call `eos` to establish the proper end-of-string character or to disable the EOS detection if the end-of-string character used by the Talker is not known.

If you call `gts` without an argument, the GPIB-RS232/485/422 returns the current Controller status, as follows:

- `CSB,0` if the GPIB-RS232/485/422 is in Standby without shadow handshaking
- `CSB,1` if the GPIB-RS232/485/422 is in Standby with shadow handshaking
- `CAC` if the GPIB-RS232/485/422 is CIC but is not in Standby—that is, it is the Active Controller
- `CIDLE` if the GPIB-RS232/485/422 is not the CIC—that is, if it is an Idle Controller

The GPIB-RS232/485/422 must be CIC when you call `gts` with an argument. If you call

`gts` with an argument and the GPIB-RS232/485/422 is not CIC, the GPIB-RS232/485/422 records the ECIC error.

See Also

`cac`.

Example 1

Go To Standby without shadow handshaking.

```
char  buffer [] = "gts 0\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

Example 2

Go To Standby with shadow handshaking.

```
char  buffer [] = "gts 1\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

Example 3

Get standby status.

```
char  buffer [] = "gts\r";
char  standbyStatus[5];
memset(standbyStatus,0,5);
viWrite(instr, buffer, strlen(buffer), &retCount);
viRead(instr, standbyStatus, 5, &retCount);
printf("Standby status is %s\n", standbyStatus);
```

id

Identify System

Type

General-use function

Syntax

id<CR>

Purpose

You can use `id` to know the revision level of the software in your GPIB-RS232/485/422.

Remarks

The identification is returned in three strings. The first string identifies the company product model. The second string is a copyright notice. The third string identifies the GPIB-RS232/485/422 firmware version.

The following example shows the current identification string at the time of this writing. The general format is as shown; however, version-specific information such as revision levels and copyright dates change as needed.



Note There are format differences between the GPIB-RS232/485/422 and GPIB-232/485CT/CV-A regarding the id string returned. For more information about this and other behavioral differences, refer to Differences Between the GPIB-232/485CT/CV-A and GPIB-RS232/485/422.

Example

```
char idQuery [] = "id\r";
char firstLine [50];
char secondLine [50];
char thirdLine [50];
memset(firstLine,0,50);
memset(secondLine,0,50);
memset(thirdLine,0,50);
viWrite (instr, idQuery, strlen(idQuery), &retCount);
viRead  (instr, firstLine, 50, &retCount);
viRead  (instr, secondLine, 50, &retCount);
```

```
viRead  (instr, thirdLine, 50, &retCount);

entire buffer contains:
       GPIB-RS232<CR><LF>
       (c)2006 National Instruments<CR><LF>
       Version 1.0.0f0<CR><LF>
```

ist

Set or Clear Individual Status Bit

Type

Parallel poll function

Syntax

is`t [bool]`<CR>

Purpose

You can use `ist` when the GPIB-RS232/485/422 participates in a parallel poll conducted by another device that is Active Controller.

Remarks

If the argument `bool` is `1`, the GPIB-RS232/485/422's individual status bit is set to 1. If the argument `bool` is `0`, the GPIB-RS232/485/422's individual status bit is cleared. The power-on default is `0`.

If you call `ist` without an argument, the GPIB-RS232/485/422 returns the value of its individual status bit.

The assignment made by `ist` remains in effect until you call `ist` again, call `onl`, or turn off the GPIB-RS232/485/422.

See Also

ppc and Parallel Polling.

Example 1

Set the ist bit to 1.

```
char  buffer [] = "ist 1\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

Example 2

Clear the ist bit to 0.

```
char  buffer [] = "ist 0\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

Example 3

Determine the ist bit state.

```
// Get GPIB-RS232 system identification
char  buffer [] = "ist\r";
char  istState[5];
memset(istState,0,5);
viWrite(instr, buffer, strlen(buffer), &retCount);
viRead(instr, istState, 5, &retCount);
printf("ist is %s\n", istState);
```

lines

Return State of GPIB Control Lines

Type

Low-level bus management function

Syntax

lines<CR>

Purpose

You can use `lines` to determine the state of the GPIB control lines.

Remarks

This command returns two numeric strings of information. The numeric strings are separated by a comma and terminated with <CR><LF>. The state of the eight GPIB control lines is returned in the first number. Each bit in this number corresponds to a GPIB control line as follows:

7	6	5	4	3	2	1	0
EOI*	ATN*	SRQ*	REN*	IFC*	NRFD*	NDAC*	DAV*

The second number contains mask bits in the same order as above, indicating which lines are actually being reported, and which are indeterminable. If a particular mask bit is 1, the corresponding bit in the first number indicates the line state. If the mask bit is `0`, the corresponding bit in the first number should be disregarded.

See Also

GPIB Basics.

Example

Determine the state of the GPIB control lines.

```
char  buffer [] = "lines\r";
char  linesState[10];
memset(linesState,0,10);
int gpibLines = 0;
viWrite(instr, buffer, strlen(buffer), &retCount);
```

```
viRead(instr, linesState, 10, &retCount);
gpibLines = atoi(linesState);
if(gpibLines & (1 << 7))
{
   printf("EOI asserted\n");
}
if(gpibLines & (1 << 6))
{
   printf("ATN asserted\n");
}
if(gpibLines & (1 << 5))
{
   printf("SRQ asserted\n");
}
if(gpibLines & (1 << 4))
{
   printf("REN asserted\n");
}
if(gpibLines & (1 << 3))
{
   printf("IFC asserted\n");
}
if(gpibLines & (1 << 2))
{
   printf("NRFD asserted\n");
}
if(gpibLines & (1 << 1))
{
   printf("NDAC asserted\n");
}
if(gpibLines & 1)
{
   printf("DAV asserted\n");
}
```

In

Check for Listening Devices

Type

Low-level bus management function

Syntax

ln `alist`<CR>

Purpose

You can use `ln` to determine whether or not there are listening devices at the specified GPIB addresses.

Remarks

The argument `alist` is a list of addresses separated by commas or spaces. There should be at least one address in the list. addresses specify the GPIB addresses of the devices you want to check.

When specifying the address parameters, a value of 255 (hex FF) can be used as a secondary address. When this special value is used, the GPIB-RS232/485/422 checks all secondary addresses for the specified primary address.

If this is the first function you call that requires GPIB Controller capability, and you have not disabled System Controller capability with `rsc`, the GPIB-RS232/485/422 sends Interface Clear (IFC*) to make itself CIC. It also asserts Remote Enable.

If you passed control to some other GPIB device, control must be passed back to you or you must send IFC* to make yourself CIC before making this call. Otherwise, the ECIC error is posted.

`ln` returns the list of addresses found to be listening on the GPIB bus. The addresses of this list are separated by commas and terminated with <CR><LF>. If none of the specified addresses were found to be listening, a <CR><LF> alone is returned.

Example

Look for devices at primary address 2, primary address 5 with all secondary addresses, primary address 3 secondary address 20, and primary address 7.

```
char  buffer [] = "ln 2,5+255,3+20,7\r";
```

```
char  listenAddresses[20];
memset(listenAddresses,0,20);
viWrite(instr, buffer, strlen(buffer), &retCount);
viRead(instr, listenAddresses, 20, &retCount);
printf("Devices found at addresses: %s\n", listenAddresses);
```

loc

Go to Local*

Type

High-level bus management function

Syntax

l`oc [alist]`<CR>

Purpose

You can use `loc` to put a GPIB device in local program mode. In this mode, you can program the device from its front panel. Because a device must usually be placed in remote program mode before it can be programmed from the GPIB, the GPIB-RS232/485/422 automatically puts the device in remote program mode. You then use `loc` to return devices to local program mode.

Remarks

The argument `alist` is a list of addresses separated by commas or spaces. Addresses specify the GPIB addresses of the devices you want to return to local mode.

If you call `loc` with `alist`, the GPIB-RS232/485/422 places the specified device(s) in local mode using the Go To Local (GTL) command.

If you call `loc` without `alist`, and the GPIB-RS232/485/422 is System Controller, the GPIB-RS232/485/422 returns all devices to local mode by unasserting REN* and asserting it again. If you call `loc` without `alist` and the GPIB-RS232/485/422 is not

System Controller, the GPIB-RS232/485/422 records the ESAC error.

If this is the first function you call that requires GPIB Controller capability, and you have not disabled System Controller capability with `rsc`, the GPIB-RS232/485/422 sends Interface Clear (IFC*) to make itself CIC. It also asserts Remote Enable.

If you passed control to some other GPIB device, control must be passed back to you or you must send IFC* to make yourself CIC before making this call. Otherwise, the ECIC error is posted.

If only the special value of 255 (hex FF) is entered as a parameter, the GPIB-RS232/485/422 configures itself for local program mode by pulsing its internal `rtl` (return to local) message. The GPIB-RS232/485/422 does not require GPIB Controller capability to configure itself for local program mode.

See Also

Status and Error Message Information.

Example 1

Put three devices in local mode.

```
char  buffer [] = "loc 6+22,4+23,7\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

Example 2

Put all devices in local mode.

```
char  buffer [] = "loc\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

Example 3

Put the GPIB-RS232/485/422 in local mode.

```
char  buffer [] = "loc 255\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

onl

Place the GPIB-RS232/485/422 Online/Offline

Type

Initialization function

Syntax

o`nl [bool]`<CR>

Purpose

You can use `onl` to disable communications between the GPIB-RS232/485/422 and the GPIB, or to reinitialize the GPIB-RS232/485/422 characteristics to their default values.

Remarks

If the argument `bool` is `1`, the GPIB-RS232/485/422 places itself online. If the argument `bool` is `0`, the GPIB-RS232/485/422 takes itself offline. By default, the GPIB-RS232/485/422 powers up online, is in the Idle Controller state, and configures itself to be the System Controller.

Placing the GPIB-RS232/485/422 offline can be thought of as disconnecting its GPIB cable from the other GPIB devices.

Placing the GPIB-RS232/485/422 online allows the GPIB-RS232/485/422 to communicate over the GPIB and restores all settings except the ones changed with the `xon` command to their power-on values.



Note Restoring all settings to their power-on values except the ones changed with the `xon` command is a difference from the GPIB-232/485/CT/CV-A. For more information about this

	and other differences, refer to Differences Between the GPIB-232/485CT/CV-A and GPIB-RS232/485/422.

If you call `onl` without an argument, the GPIB-RS232/485/422 returns the current state, which is `0` if the GPIB-RS232/485/422 is offline and `1` if the GPIB-RS232/485/422 is online.

See Also

S Mode Default Settings and Related Functions.

Example 1

Put the GPIB-RS232/485/422 online and restore its power on settings.

```
char  buffer [] = "onl 1\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

Example 2

Put the GPIB-RS232/485/422 offline to prevent it from communicating on GPIB.

```
char  buffer [] = "onl 0\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

pct

Pass Control

Type

Low-level bus management function

Syntax

pct `addr`<CR>

Purpose

You can use `pct` to pass Controller-In-Charge (CIC) authority from the GPIB-RS232/485/422 to some other GPIB device.

Remarks

The argument `addr` is the address of the device you want to pass control to. `addr` consists of a primary address and an optional secondary address.

`pct` passes CIC authority from the GPIB-RS232/485/422 to the device specified by `addr`. The GPIB-RS232/485/422 automatically goes to Idle Controller State. It is assumed that the target device has Controller capability.

If you call `pct` with an argument and the GPIB-RS232/485/422 is not CIC, it records the ECIC error.

If you call `pct` without an argument, the GPIB-RS232/485/422 records the EARG error.

Example

Pass control to a device with primary address 7 and secondary address 18.

```
char  buffer [] = "pct 7+18\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

ppc

Parallel Poll Configure

Type

Parallel poll function

Syntax

ppc `addr,ppr,s [addr,ppr,s]...`<CR>

Purpose

You can use `ppc` to configure specified GPIB devices to respond to parallel polls in a certain manner.

Remarks

The argument `addr` specifies the GPIB address of the device to be enabled for parallel polls. `addr` consists of a primary address and an optional secondary address.

The argument `ppr` is an integer string between `1` and `8` that specifies the data line on which to respond.

The argument `s` is either `0` or `1` and is interpreted along with the value of the device's individual status bit to determine whether to drive the line true or false.

Each group of `addr,ppr,s` can be separated by either a comma or space, just as any list of arguments.

If this is the first function you call that requires GPIB Controller capability, and you have not disabled System Controller capability with `rsc`, the GPIB-RS232/485/422 sends Interface Clear (IFC*) to make itself CIC. It also asserts Remote Enable.

If you passed control to some other GPIB device, control must be passed back to you or you must send IFC* to make yourself CIC before making this call. Otherwise, the ECIC error is posted.

The GPIB-RS232/485/422 takes the arguments `ppr` and `s` and constructs the appropriate parallel poll enable (PPE) message for each addr specified.

If you call `ppc` without an argument, the GPIB-RS232/485/422 records the EARG error.

If only one `addr,ppr,s` is specified, and `addr` has the special value of 255 (hex FF), the GPIB-RS232/485/422 configures itself for parallel polls. To do this, the GPIB-RS232/

485/422 must be using IEEE 488 Parallel Poll (PP) interface function subset PP2. You can do this by setting the PP2 option using the `conf` function. If the PP2 option is not set, the GPIB-RS232/485/422 records the ECAP error. The GPIB-RS232/485/422 does not require GPIB Controller capability to configure itself for parallel polls.

See Also

`conf`, `ist`, `ppu`, `rpp`, and Parallel Polling.

Example 1

Configure two devices for parallel poll.

```
char  buffer [] = "ppc 18+23,8,0 23+10,7,1\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

Example 2

Configure the GPIB-RS232/485/422 to respond to parallel polls on data line 7 if its individual status bit is 1.

```
char  buffer [] = "ppc 255,7,1\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

ppu

Parallel Poll Unconfigure

Type

Parallel poll function

Syntax

ppu `[alist]`<CR>

Purpose

You can use `ppu` if you are performing parallel polls and you want to prevent certain GPIB devices from responding.

Remarks

The argument `alist` is a list of addresses separated by commas or spaces. Addresses specify the GPIB addresses of the devices to be disabled from parallel polls.

If you call `ppu` with `alist`, the GPIB-RS232/485/422 unconfigures from parallel polls only those devices specified in `alist`. If you call `ppu` without `alist`, the GPIB-RS232/485/422 unconfigures all devices from parallel polls.

If this is the first function you call that requires GPIB Controller capability, and you have not disabled System Controller capability with `rsc`, the GPIB-RS232/485/422 sends Interface Clear (IFC*) to make itself CIC. It also asserts Remote Enable.

If you passed control to some other GPIB device, control must be passed back to you or you must send IFC* to make yourself CIC before making this call. Otherwise, the ECIC error is posted.

If only `addr` is specified and it has the special value of 255 (hex FF), the GPIB-RS232/485/422 disables itself from responding to parallel polls. To do this, the GPIB-RS232/485/422 must be using IEEE 488 Parallel Poll (PP) interface function subset PP2. You can do this by setting the PP2 option using the `conf` function. If the PP2 option is not set, the GPIB-RS232/485/422 records the ECAP error. The GPIB-RS232/485/422 does not require GPIB Controller capability to disable itself from responding to parallel polls.

See Also

`conf`, `ist`, `ppc`, `rpp`, and Parallel Polling.

Example 1

Send the PPD command to the device at primary address 14.

```
char  buffer [] = "ppu 14\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

Example 2

Send the PPD command to all devices.

```
char  buffer [] = "ppu\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

Example 3

Disable the GPIB-RS232/485/422 from responding to parallel polls.

```
char  buffer [] = "ppu 255\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

rd

Read Data*

Type

I/O function

Syntax

rd `#count [addr]`<CR>

Purpose

You can use `rd` to read data from the GPIB.

Remarks

The argument `count` is a numeric string preceded by a number sign (`#`). `count`

specifies the number of GPIB data bytes to read and can range from 1 to 4294967295.

The argument `addr` specifies the GPIB address of the device to be addressed as a Talker. `Addr` consists of a primary address and an optional secondary address.

The GPIB-RS232/485/422 reads data from the GPIB until one of the following events occurs:

- The GPIB-RS232/485/422 successfully transfers all data.
- The GPIB END message is received with a data byte.
- The EOS byte is received.
- The I/O time limit is exceeded. The EABO error is recorded.
- The GPIB-RS232/485/422 receives a Device Clear. The EABO error is recorded.
- The `addr` argument is specified and the requested GPIB addressing bytes cannot be sent. The EBUS error is recorded.

Because you might not know the number of bytes actually read from the GPIB, the GPIB-RS232/485/422 returns the received GPIB data to you in the following manner. First, the GPIB-RS232/485/422 returns all bytes it read from the GPIB. Next, if the number of bytes read is less than `count`, the GPIB-RS232/485/422 sends null bytes (decimal 0) until the total number of bytes returned to you matches the number specified in count. Finally, it returns a numeric string representing the number of bytes it actually read from the GPIB.

For example, if you send the GPIB-RS232/485/422 the programming message "`rd #1 0`"<CR>, it reads data from the GPIB until it receives 10 bytes of data, the END message, or an `eos` byte. If the GPIB-RS232/485/422 received END with the fourth data byte, it would return the 4 data bytes. Then it would send 6 null bytes followed by an ASCII 4 and <CR><LF>.

You should always read back `count` bytes of data from the serial port, then look at the numeric string to determine how many bytes were read from the GPIB.

If you call `rd` with the `addr` argument, the GPIB-RS232/485/422 must be CIC to perform the addressing. If this is the first function you call that requires GPIB Controller capability, and you have not disabled System Controller capability with `rs c`, the GPIB-RS232/485/422 sends Interface Clear (IFC*) to make itself CIC. It also asserts Remote Enable.

If you call `rd` with the `addr` argument, and you previously passed control to some other GPIB device, control must be passed back to you or you must send IFC* to make yourself CIC before making this call. Otherwise, the ECIC error is recorded.

When performing the addressing for a specified `addr`, the GPIB-RS232/485/422 sends out its own listen address as well as the talk address of the specified device. It then places itself in Standby Controller state with ATN* unasserted, and remains there after the read operation is complete.

If the `addr` argument is not specified, the GPIB-RS232/485/422 assumes it has already been addressed to listen by the Controller. If the GPIB-RS232/485/422 is the Controller, and did not address itself to listen before calling `rd`, the EADR error is recorded and no data bytes are transferred.

If you call `rd` without an argument, the GPIB-RS232/485/422 records EARG.

See Also

`eos`, `eot`, and `tmo`.

Example

Read up to 10 bytes from the GPIB device at address 3.

```
char  cmdBuffer [] = "rd #10 3\r";
char  rdBuffer[11];
char  countBuffer[5];
int count = 0;
memset(rdBuffer,0,11);
memset(countBuffer,0,5);
//Disable Termination Character to allow all NULL
//characters to be read from the serial port
viSetAttribute (instr, VI_ATTR_TERMCHAR_EN, VI_FALSE);
viSetAttribute (instr, VI_ATTR_ASRL_END_IN, VI_ASRL_END_NONE);
viWrite(instr, cmdBuffer, strlen(cmdBuffer), &retCount);
viRead(instr, rdBuffer, 10, &retCount);
//Re-enable Termination Character
viSetAttribute (instr, VI_ATTR_TERMCHAR_EN, VI_TRUE);
viSetAttribute (instr, VI_ATTR_ASRL_END_IN, VI_ASRL_END_TERMCHAR);
viRead(instr, countBuffer, 4, &retCount);
```

```
count = atoi(countBuffer);
printf("S Mode read %d bytes from device at PAD 3.\n", count);
```

rpp

Request (Conduct) a Parallel Poll

Type

Parallel poll function

Syntax

rpp<CR>

Purpose

You can use `rpp` to conduct a parallel poll to obtain information from several GPIB devices at the same time.

Remarks

`rpp` causes the GPIB-RS232/485/422 to conduct a parallel poll of previously configured devices by sending the IDY* (Identify) message (ATN* and EOI* both asserted) and reading the response from the GPIB data lines. The GPIB-RS232/485/422 pulses the IDY* message for greater than or equal to 2 µs and expects valid responses within that time. It remains Active Controller after pulsing the IDY* message.

The GPIB-RS232/485/422 returns the Parallel Poll Response (PPR) following the poll in the form of a numeric string representing the decimal value of the response.

If this is the first function you call that requires GPIB Controller capability, and you have not disabled System Controller capability with `rsc`, the GPIB-RS232/485/422 sends Interface Clear (IFC*) to make itself CIC. It also asserts Remote Enable.

If you passed control to some other GPIB device, control must be passed back to you

or you must send IFC* to make yourself CIC before making this call. Otherwise, the ECIC error is posted.

See Also

ist, ppc, ppu, and Parallel Polling.

Example

```
char  ppcBuffer [] = "ppc 13,1,0 15,3,0\r";
char  rppBuffer [] = "rpp\r";
char  rppResponseBuffer[6];
int   rppResponse = 0;
memset(rppResponseBuffer,0,6);
viWrite(instr, ppcBuffer, strlen(ppcBuffer), &retCount);
viWrite(instr, rppBuffer, strlen(rppBuffer), &retCount);
viRead(instr, rppResponseBuffer, 6, &retCount);
rppResponse = atoi(rppResponseBuffer);
if(rppResponse & 1)
{
printf("Device at address 13 responded positive to parallel poll\n");
}
if(rppResponse & 4)
{
printf("Device at address 15 responded positive to parallel poll\n");
}
```

rsc

Request System Control

Type

Initialization function

Syntax

rsc [bool]<CR>

Purpose

You can use `rsc` to change which device in your GPIB system is System Controller.

Remarks

If the argument `bool` is `1`, the GPIB-RS232/485/422 configures itself to be the GPIB System Controller. If the argument `bool` is `0`, the GPIB-RS232/485/422 unconfigures itself to be the System Controller.

If you call `rsc` without an argument, the GPIB-RS232/485/422 returns its System Controller status, which is `0` if the GPIB-RS232/485/422 is not currently System Controller or `1` if the GPIB-RS232/485/422 is System Controller.

As System Controller, the GPIB-RS232/485/422 can send the Interface Clear (IFC*) and Remote Enable (REN*) messages to GPIB devices. If some other Controller asserts IFC*, the GPIB-RS232/485/422 can respond only if it is not configured as System Controller.

In most applications, the GPIB-RS232/485/422 is System Controller. In some applications, the GPIB-RS232/485/422 is never System Controller. In either case, `rsc` is used only if the computer will not be System Controller while the program executes. The IEEE 488 standard does not specifically allow schemes in which System Control can be passed from one device to another; however, `rsc` could be used in such a scheme. The GPIB-RS232/485/422 configures itself to be System Controller at power-on. The assignment made by `rsc` remains in effect until you call `rsc` again, call `onl`, or turn off the GPIB-RS232/485/422.

See Also

`sic` and `sre`.

Example 1

Enable the GPIB-RS232/485/422 to be the System Controller.

```
char  buffer [] = "rsc 1\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

Example 2

Disable the System Controller.

```
char  buffer [] = "rsc 0\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

Example 3

Determine if the GPIB-RS232/485/422 is the System Controller.

```
char  buffer [] = "rsc\r";
char rscString[4];
viWrite(instr, buffer, strlen(buffer), &retCount);
viRead(instr, rscString, 4, &retCount);
if(rscString[0] == '0')
{
   printf("S Mode is not system controller\n");
}
else if (rscString[0] == '1')
{
   printf("S Mode is system controller\n");
}
```

rsp

Request (Conduct) a Serial Poll

Type

Serial poll function

Syntax

rsp `alist`<CR>

Purpose

You can use `rsp` to conduct a serial poll to obtain device-specific status information from one or more GPIB devices.

Remarks

The argument `alist` is a list of addresses separated by commas or spaces. Addresses are device addresses that specify the GPIB addresses you want to poll.

`rsp` serial polls the specified devices to obtain their status bytes. If bit 6 (hex 40 or RQS bit) of a device's response is set, its status response is positive—that is, that device is requesting service. Before `rsp` completes, all devices are unaddressed.

The interpretation of each device response, other than the RQS bit, is device specific. For example, the polled device might set a particular bit in the response byte to indicate that it has data to transfer, and another bit to indicate a need for reprogramming. Consult the device documentation for interpretation of the response byte.

Each device serial poll response byte is returned as a numeric string giving the decimal value of the byte, followed by <CR> and <LF>. If a device does not respond in the timeout period, the GPIB-RS232/485/422 returns the string `-1` and records the EABO error. The time limit is set to 0.1 second unless you called `tmo` to change it. Each response corresponds directly to an address you specify. So, the number of response lines, including `-1`, exactly matches the number of addresses you specify.

If this is the first function you call that requires GPIB Controller capability, and you have not disabled System Controller capability with `rsc`, the GPIB-RS232/485/422 sends Interface Clear (IFC*) to make itself CIC. It also asserts Remote Enable.

If you passed control to some other GPIB device, control must be passed back to you or you must send IFC* to make yourself CIC before making this call. Otherwise, the ECIC error is posted.

If you call `rsp` without an argument, the GPIB-RS232/485/422 records the EARG error.

See Also

tmo.

Example

Serial poll three devices.

```
char  buffer [] = "rsp 1+28,5,9\r";
char response1[6];
char response2[6];
char response3[6];
memset(response1,0,6);
memset(response2,0,6);
memset(response3,0,6);
viWrite(instr, buffer, strlen(buffer), &retCount);
viRead(instr, response1, 5, &retCount);
viRead(instr, response2, 5, &retCount);
viRead(instr, response3, 5, &retCount);
printf("Device at PAD 1, SAD 28 reponded %s",response1);
printf("Device at PAD 5 responded %s", response2);
printf("Device at PAD 9 responded %s", response3);
```

rsv

Request Service/Set or Change Serial Poll Status Byte

Type

Serial poll function

Syntax

rsv `[spbyte]`<CR>

Purpose

You can use `rsv` if the GPIB-RS232/485/422 is not the GPIB Controller and you want to

request service from the Controller using the Service Request (SRQ*) signal. The GPIB-RS232/485/422 provides a user-defined status byte when the Controller serial polls it.

Remarks

The argument `spbyte` is a numeric string specifying the decimal value of the new GPIB-RS232/485/422 serial poll response byte.

The serial poll response byte is the status byte the GPIB-RS232/485/422 provides when serial polled by another device that is CIC. If bit 6 (hex 40 or RQS bit) is also set, the GPIB-RS232/485/422 requests service by asserting the SRQ* line.

If you call `rsv` without an argument, the GPIB-RS232/485/422 returns a numeric string containing the decimal value of its serial poll status byte.

The assignment made by `rsv` remains in effect until you call `rsv` again, call `onl`, or turn off the GPIB-RS232/485/422.

Example 1

Request service with serial poll response of 0x46.

```
char  buffer [] = "rsv \\x46\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

Example 2

Query the current serial poll status byte.

```
char  buffer [] = "rsv\r";
char rsvString[6];
memset(rsvString,0,6);
viWrite(instr, buffer, strlen(buffer), &retCount);
viRead(instr, rsvString, 6, &retCount);
printf("S Mode serial poll response byte is: 0x%X\n",atoi(rsvString));
```

sic

Send Interface Clear

Type

Low-level bus management function

Syntax

sic<CR>

Purpose

You can use `sic` if the initialization, I/O, or high-level bus management functions do not meet the needs of your device, and you want to have more precise control over the GPIB. `sic` makes the GPIB-RS232/485/422 CIC and initializes the GPIB. `sic` is not a function you need to use frequently, because in most cases the first I/O or high-level bus management function you call does the same things automatically.

Remarks

The `sic` command asserts the GPIB interface clear (IFC*) signal for at least 100 µs, then unasserts IFC*. Asserting IFC* unaddresses all devices on the bus and makes the GPIB-RS232/485/422 CIC.

The IFC* signal resets only the GPIB interface functions of bus devices and not the internal device functions. Device functions are reset with the `clr` programming message. To determine the effect of these messages, consult the device documentation.

The GPIB-RS232/485/422 records the ESAC error if you have disabled its System Controller capability with the `rsc` function.

See Also

`clr`, GPIB Basics, and Differences Between the GPIB-232/485CT/CV-A and GPIB-RS232/485/422.

Example 1

Send interface clear for 100 µs.

```
char  buffer [] = "sic\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

spign

Ignore Serial Port Errors

Type

Serial port function

Syntax

spi`gn [bool]`<CR>

Purpose

You can use `spign` at the beginning of your program to change the effect that serial port errors have on how the GPIB-RS232/485/422 processes programming messages and data. This function tells the GPIB-RS232/485/422 to ignore or not to ignore the occurrence of serial port errors. By default, serial port errors are ignored.

Remarks

If the argument `bool` is `0`, the GPIB-RS232/485/422 does not ignore serial port errors. In this case, the GPIB-RS232/485/422 does not execute programming messages that contain serial port errors. Also, if a serial port error occurs with any byte contained in a `cmd` or `wrt` data string, the GPIB-RS232/485/422 discards that data byte and all remaining bytes in the string.

The serial port errors include parity, overrun, framing, and overflow errors. A list of serial port errors is in Status and Error Message Information.

If the argument `bool` is `1`, the GPIB-RS232/485/422 executes all programming messages and sends all data, even if serial port errors occur as the messages and data bytes are received. No serial error code is reported by the `stat` function.

If you call `spign` without an argument, the GPIB-RS232/485/422 returns the current setting.

The assignment made by `spign` remains in effect until you call `spign` again, call `on 1`, or turn off the GPIB-RS232/485/422.

See Also

`cmd` and `wrt`.

Example 1

Configure the GPIB-RS232/485/422 not to execute programming messages or process data that contains serial port errors.

```
char  buffer [] = "spign 0\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

Example 2

Configure the GPIB-RS232/485/422 to execute programming messages and send all data, even if serial port errors occur.

```
char  buffer [] = "spign 1\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

sre

Set (or Clear) Remote Enable

Type

Low-level bus management function

Syntax

sre `[bool]`<CR>

Purpose

Use `sre` to turn the Remote Enable signal on and off. In some cases, the first I/O or high-level bus management function you call sets remote enable automatically.

Remarks

If the argument `bool` is `1`, the GPIB-RS232/485/422 asserts the Remote Enable (REN*) signal. If the argument `bool` is `0`, the GPIB-RS232/485/422 unasserts REN*.

Many GPIB devices have a remote program mode and a local program mode. It is usually necessary to place devices in remote mode before programming them from the GPIB. A device enters the remote mode when the REN* line is asserted and the device receives its listen address.

Use `cmd` to send a device its listen address after using `sre`. Use `loc` to return the device to local program mode.

If you call `sre` with an argument and the GPIB-RS232/485/422 is not System Controller, the ESAC error is recorded.

If you call `sre` without an argument, the GPIB-RS232/485/422 returns its current remote status: `1` = remote, `0` = local.

See Also

`cmd`, `loc`, and `rsc`.

Example 1

Assert REN*.

```
char  buffer [] = "sre 1\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

Example 2

Unassert REN*.

```
char  buffer [] = "sre 0\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

stat

Return GPIB-RS232/485/422 Status

Type

General-use function

Syntax

stat [[c] n]<CR>

or

stat [c] s<CR>

or

stat [c] n s<CR>

Purpose

You can use `stat` to obtain the GPIB-RS232/485/422 status to see if certain conditions are currently present. `stat` is used most often to see if the previous operation resulted in an error.

Remarks

You should use `stat` frequently in the early stages of your program development when the responses of your GPIB devices are likely to be unpredictable.

The GPIB-RS232/485/422 responds with status information in a form depending on the mode or combination of modes you choose. `n` indicates that the status information is returned as numeric strings. `s` indicates that the status information is returned in symbolic format—that is, as mnemonic strings. Status bits are separated by a comma. Normally, you use `s` only when you are debugging your code and want to print the mnemonic for each piece of status information. `c` specifies that the status is returned after each programming message, eliminating the need to call `stat` after each programming message.

If you call `stat` without an argument, continuous status reporting is disabled.

The status information returned by the GPIB-RS232/485/422 contains four pieces of information: the GPIB-RS232/485/422 status, a GPIB error code, a serial error code, and a count. A <CR><LF> follows each piece of the response.

Status represents a combination of GPIB-RS232/485/422 conditions. Inside the GPIB-RS232/485/422, status is stored as a 16-bit integer. Each bit in the integer represents a single condition. A bit value of 1 indicates that the corresponding condition is in effect. A bit value of 0 indicates that the condition is not in effect. Because more than one GPIB-RS232/485/422 condition can exist at one time, more than one bit can be set in status. The highest order bit of status, also called the sign bit, is set when the GPIB-RS232/485/422 detects either a GPIB error or a serial port error. When status is negative, an error condition exists.

The following table lists the values and descriptions of the GPIB status conditions that the `stat` function might return.

S Mode GPIB Status Conditions `stat` Returns

Numeric Value (n)	Symbolic Value (s)	Description	Bit
–32768	ERR	Error detected	15
16384	TIMO	Timeout	14
8192	END	EOI* or EOS detected	13
4096	SRQI	SRQ* detected while CIC	12
2048	—	Reserved	11
1024	—	Reserved	10
512	—	Reserved	9
256	CMPL	Operation completed	8
128	LOK	Lockout state	7
64	REM	Remote state	6
32	CIC	Controller-In-Charge	5
16	ATN	Attention asserted	4
8	TACS	Talker active	3
4	LACS	Listener active	2
2	DTAS	Device trigger active state	1
1	DCAS	Device clear active state	0

The GPIB error code represents a single GPIB error condition present.

The serial error code represents a single serial port error condition present.

`count` is the number of bytes transferred over the GPIB by the last `rd`, `wrt`, or `cmd` function.

The following table lists values and descriptions of GPIB error conditions that the `stat` function might return.

S Mode GPIB Error Conditions `stat` Returns

Numeric Value (n)	Symbolic Value (s)	Description
0	NGER	No GPIB error condition to report
1	ECIC	Command requires GPIB-RS232/485/422 to be CIC
2	ENOL	Write detected no Listeners
3	EADR	GPIB-RS232/485/422 not addressed correctly
4	EARG	Invalid argument or arguments
5	ESAC	Command requires GPIB-RS232/485/422 to be System Controller
6	EABO	I/O operation aborted
7-10	—	Reserved
11	ECAP	No capability for operation
12-13	—	Reserved
14	EBUS	Command bytes could not be sent
15-16	—	Reserved
17	ECMD	Unrecognized command

The following table lists the serial port error conditions that the `stat` function might return.

S Mode Serial Port Error Conditions `stat` Returns

Numeric Value (n)	Symbolic Value (s)	Description
0	NSER	No serial port error condition to report
1	EPAR	Serial port parity error
2	EORN	Serial port overrun error
3	EOFL	Serial port receive buffer overflow
4	EFRM	Serial port framing error

A detailed description of the conditions under which each bit in status is set or cleared

is in Status and Error Message Information.

In general, the GPIB-RS232/485/422 updates the first three status variables at the end of each programming message. It updates the fourth status variable, `count`, after a `cmd`, `rd`, or `wrt` function. The errors reported correspond to the previous programming message. For example, if you call `wrt` and then `stat s`, any errors returned to you correspond to errors in the `wrt` programming message, not `stat`. However, if status is returned in continuous mode, the status information corresponds to the current programming message. For example, suppose you called `stat c s` to set up continuous status reporting. After reading the status information returned from the `stat` call, you call `wrt`. The GPIB-RS232/485/422 then returns the status information that corresponds to the `wrt` message.

To begin continuous status reporting, send the `stat c s`, `stat c n`, or `stat c n s` programming message. Status information is immediately returned indicating the current status conditions. When you call `stat` with both `s` and `n`, the numeric status is always returned first.

Notice that when you send several programming messages to the GPIB-RS232/485/422, it buffers them and processes each one without any delay in between. However, if you enable continuous status reporting and check the status of each programming message before sending the next, the GPIB-RS232/485/422 waits for each subsequent programming message to arrive at the serial port before processing it. This slows down the overall performance of your program. If speed is a primary concern, disable continuous status reporting.

The continuous status setting remains in effect until you call `stat` again, call `onl`, or turn off the GPIB-RS232/485/422.

Example 1

Get the GPIB-RS232/485/422 status as numeric values.

```
char  statusValueBuffer[10];
int   statusValue;
char  gpibErrorBuffer[10];
int   gpibError;
char  serialErrorBuffer[10];
```

```
int   serialError;
char  countBuffer[10];
int   count;
char  buffer [] = "stat n\r";

memset(statusValueBuffer,0,10);
memset(gpibErrorBuffer,0,10);
memset(serialErrorBuffer,0,10);
memset(countBuffer,0,10);

viWrite(instr, buffer, strlen(buffer), &retCount);
viRead(instr, statusValueBuffer, 10, &retCount);

statusValue = atoi(statusValueBuffer);

if(statusValue & (1 << 15))
{
   printf("Error detected\n");
}
if(statusValue & (1 << 14))
{
   printf("Timeout\n");
}
if(statusValue & (1 << 13))
{
   printf("EOI* or EOS detected\n");
}
if(statusValue & (1 << 12))
{
   printf("SRQ* detected while CIC\n");
}
if(statusValue & (1 << 8))
{
   printf("Operation Complete\n");
}
if(statusValue & (1 << 7))
{
   printf("Lockout State\n");
}
if(statusValue & (1 << 6))
{
   printf("Remote State\n");
}
if(statusValue & (1 << 5))
{
   printf("Controller-In-Charge\n");
```

```
}
if(statusValue & (1 << 4))
{
   printf("Attention Asserted\n");
}
if(statusValue & (1 << 3))
{
   printf("Talker Active\n");
}
if(statusValue & (1 << 2))
{
   printf("Listener Active\n");
}
if(statusValue & (1 << 1))
{
   printf("Device Trigger Active State\n");
}
if(statusValue & 1)
{
   printf("Device Clear Active State\n");
}


viRead(instr, gpibErrorBuffer, 10, &retCount);
gpibError = atoi(gpibErrorBuffer);
switch(gpibError)
{
   case 0:
      printf("No GPIB error condition\n");
      break;
   case 1:
      printf("Command requires S Mode to be CIC\n");
      break;
   case 2:
      printf("No listeners detected\n");
      break;
   case 3:
      printf("S Mode not addressed correctly\n");
      break;
   case 4:
      printf("Invalid argument\n");
      break;
   case 5:
      printf("Command requires S Mode to be System Controller\n");
      break;
   case 6:
```

```
      printf("I/O operation aborted\n");
      break;
   case 11:
      printf("S Mode has no capability for that operation\n");
      break;
   case 14:
      printf("Command bytes could not be sent\n");
      break;
   case 17:
      printf("Unrecognized command\n");
      break;
}

viRead(instr, serialErrorBuffer, 10, &retCount);
serialError = atoi(serialErrorBuffer);
switch(serialError)
{
   case 0:
      printf("No serial port error\n");
      break;
   case 1:
      printf("Serial port parity error\n");
      break;
   case 2:
      printf("Serial port overrun error \n");
      break;
   case 3:
      printf("Serial port receive buffer overflow error \n");
      break;
   case 4:
      printf("Serial port framing error\n");
      break;
}

viRead(instr, countBuffer, 10, &retCount);
count = atoi(countBuffer);
printf("Count is %d\n",count);
```

Example 2

Get the GPIB-RS232/485/422 status as symbolic values.

```
char  statusValueBuffer[30];
char  gpibErrorBuffer[10];
```

```
char  serialErrorBuffer[10];
char  countBuffer[10];
int   count;

char  buffer [] = "stat s\r";

memset(statusValueBuffer,0,30);
memset(gpibErrorBuffer,0,10);
memset(serialErrorBuffer,0,10);
memset(countBuffer,0,10);

viWrite(instr, buffer, strlen(buffer), &retCount);

viRead(instr, statusValueBuffer, 30, &retCount);
viRead(instr, gpibErrorBuffer, 30, &retCount);
viRead(instr, serialErrorBuffer, 30, &retCount);
viRead(instr, countBuffer, 30, &retCount);
count = atoi(countBuffer);
```

Example 3

The following list illustrates what appears on the screen when you are programming the GPIB-RS232/485/422 from a terminal. GPIB-RS232/485/422 responses are in bold text. The statements in parentheses are comments.

```
stat c s n (Enable continuous status reporting.)
344
0
0
3
CMPL,REM,ATN,TACS ( Status returned. )
NGER
NSER
3
wrt 10
ABCDE ( Write the string ABCDE. )
296 ( to device 10. )
0 ( Status returned. )
0
5
CMPL,CIC,TACS
NGER
NSER
```

```
5
```

tmo

Change or Disable Time Limit

Type

Initialization function

Syntax

tmo [timeio][,timesp]<CR>

Purpose

You can use tmo at the beginning of your program to change the time limits in effect on the GPIB-RS232/485/422. The time limits prevent the GPIB-RS232/485/422 from waiting indefinitely for critical events to occur.

Remarks

The arguments timeio and timesp are numeric strings. timeio specifies the amount of time in seconds the GPIB-RS232/485/422 waits for an I/O operation (rd, wrt, cmd) or the wait function to complete. timesp specifies the amount of time in seconds each device is given to respond to a serial poll. The power-on timeouts are 10 seconds for timeio and 0.1 second for timesp.

timeio and timesp can be any decimal number between .00001 and 3600, which corresponds to time limits between 10 µs and 1 hour. For example, 10,.1 specifies a time of 10 seconds for I/O operations and 1/10 of a second for serial poll responses. timeio and timesp can also be 0, which disables either timeout accordingly. Neither timeio nor timesp can contain commas.



Note There are minor behavioral differences between the GPIB-RS232/485/422 and GPIB-232/485CT/CV-A regarding this function. For more information about this and other behavioral

	differences, refer to Differences Between the GPIB-232/485CT/CV-A and GPIB-RS232/485/422.

The `timeio` time limit is in effect for the `cmd`, `rd`, and `wrt` functions. If the GPIB-RS232/485/422 cannot complete any of these functions within the period of time set by `timeio`, it aborts the function and records the EABO error. Bytes transferred before the timeout are not affected. The `timeio` time limit is also the maximum amount of time the wait function waits when you call it with the TIMO bit set in the wait mask.

The `timesp` time limit is in effect only for the `rsp` function. If a polled device fails to respond within the amount of time indicated by `timesp`, the GPIB-RS232/485/422 reports an error.

To change only the timeout value for serial polls, a comma must precede the serial poll timeout value.

If you call `tmo` without an argument, the GPIB-RS232/485/422 returns a numeric string representing the current timeout settings. It records the EARG error if you specify a time value outside the range of .00001 to 3600 seconds.

The assignment made by `tmo` remains in effect until you call `tmo` again, call `onl`, or turn off the GPIB-RS232/485/422.

See Also

`rsp`.

Example 1

Set the timeout for I/O operations to 30 seconds.

```
char  buffer [] = "tmo 30\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

Example 2

Set the serial poll timeout for one second, while leaving the I/O timeout unchanged.

```
char  buffer [] = "tmo ,1\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

Example 3

Retrieve the current timeout settings.

```
char  buffer [] = "tmo\r";
char tmoBuffer[20];
viWrite(instr, buffer, strlen(buffer), &retCount);
viRead(instr,tmoBuffer,20,&retCount);
memset(tmoBuffer,0,20);
printf("timeout settings are %s\n", tmoBuffer);
```

trg

Trigger Selected Device(s)*

Type

High-level bus management function

Syntax

tr`g alist`<CR>

Purpose

You can use `trg` to trigger the specified GPIB devices. The documentation for each GPIB device explains when you should trigger it and what effect the trigger has.

Remarks

The argument `alist` is a list of addresses separated by commas or spaces. Addresses are device addresses that specify the GPIB addresses you want to trigger.

If you call `trg` without an argument, the EARG error is posted.

If this is the first function you call that requires GPIB Controller capability, and you have not disabled System Controller capability with `rsc`, the GPIB-RS232/485/422 sends Interface Clear (IFC*) to make itself CIC. It also asserts Remote Enable.

If you passed control to some other GPIB device, control must be passed back to you or you must send IFC* to make yourself CIC before making this call. Otherwise, the ECIC error is posted.

Example

Trigger three devices.

```
char  buffer [] = "trg 2+10,4,5+7\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

wait

Wait for Selected Event

Type

General-use function

Syntax

wa`it mask`<CR>

Purpose

You can use `wait` to monitor selected GPIB events and to delay further GPIB-RS232/485/422 activity until any events occur.

Remarks

The argument `mask` is a numeric string that specifies the events to wait for. The numeric string represents a bit mask containing a subset of the same bit assignments

as the status word described in the `stat` function. Each bit is set to wait or cleared not to wait for the corresponding event to occur. The numeric string can be expressed as decimal, octal, or hexadecimal.

After receiving the wait programming message, the GPIB-RS232/485/422 monitors GPIB activity. When any event corresponding to the bits set in `mask` occurs, the GPIB-RS232/485/422 returns status information indicating its current status. If continuous status reporting has been enabled, status is reported in the requested format. If continuous status has not been enabled, status is returned in numeric format.

You can use `wait`, for example, to wait until a device requests service before you perform a serial poll. In this case, you send the wait programming message with `mask=4096`, and wait for status information to be returned. You then check that status to see if the SRQI bit is set in the returned status indicators.

To prevent the GPIB-RS232/485/422 from waiting indefinitely for SRQ* to be asserted, set the SRQI and TIMO bits by setting the mask equal to 20480 (16384 + 4096). This causes the wait to terminate either on SRQI or TIMO, whichever occurs first.

The following table lists the `wait mask` values.

Wait Mask Values

Decimal Value	Mnemonic	Description	Hex Value	Bit
—	—	Reserved	—	15
16384	TIMO	Timeout	4000	14
8192	END	EOI* or EOS detected	2000	13
4096	SRQI	SRQ* detected while CIC	1000	12
—	—	Reserved	—	11
—	—	Reserved	—	10
—	—	Reserved	—	9
—	—	Reserved	—	8
128	LOK	Lockout state	80	7
64	REM	Remote state	40	6

32	CIC	Controller-In-Charge	20	5
16	ATN	Attention asserted	10	4
8	TACS	Talker active	8	3
4	LACS	Listener active	4	2
2	DTAS	Device trigger state	2	1
1	DCAS	Device clear state	1	0

If `mask=0`, the function completes immediately and returns the current status.

If the TIMO bit is `0` or the `timeio` time limit is set to `0` by `tmo`, timeouts for this function are disabled. You should disable timeouts only when you are sure that the selected event occurs. Otherwise, the GPIB-RS232/485/422 waits indefinitely for the event to occur.

If you call `wait` without an argument, the GPIB-RS232/485/422 records the EARG error.

See Also

`stat` and `tmo`.

Example 1

Wait for SRQI or TIMO to occur.

```
char  statusBuffer[10];
char  gpibErrorBuffer[10];
char  serialErrorBuffer[10];
char  countBuffer[10];

char  waitCommand [] = "wait \\x5000\r";
int   waitStatus = 0;

memset(tmoBuffer,0,10);
memset(tmoBuffer,0,10);
memset(tmoBuffer,0,10);
memset(tmoBuffer,0,10);
```

```
viWrite(instr, waitCommand, strlen(waitCommand), &retCount);
viRead(instr, statusBuffer, 10, &retCount);
viRead(instr, gpibErrorBuffer, 10, &retCount);
viRead(instr, serialErrorBuffer, 10, &retCount);
viRead(instr, countBuffer, 10, &retCount);

waitStatus = atoi(statusBuffer);

if(waitStatus & 4096)
{
   printf("SRQI detected\n");
}
if(waitStatus & 16384)
{
   printf("wait command timed out\n");
}
```

Example 2

Wait indefinitely to become LACS.

```
char  statusBuffer[10];
char  gpibErrorBuffer[10];
char  serialErrorBuffer[10];
char  countBuffer[10];
char  waitCommand [] = "wait 4\r";
int   waitStatus = 0;

memset(tmoBuffer,0,10);
memset(tmoBuffer,0,10);
memset(tmoBuffer,0,10);
memset(tmoBuffer,0,10);

viWrite(instr, waitCommand, strlen(waitCommand), &retCount);
viRead(instr, statusBuffer, 10, &retCount);
viRead(instr, gpibErrorBuffer, 10, &retCount);
viRead(instr, serialErrorBuffer, 10, &retCount);
viRead(instr, countBuffer, 10, &retCount);

waitStatus = atoi(statusBuffer);

if(waitStatus & 4)
{
```

```
   printf("LACS detected\n");
}
```

wrt

Write Data*

Type

I/O function

Syntax

wr`t [#count][alist]`<LF>
`data`<CR>

Purpose

You can use `wrt` to send data over the GPIB.

Remarks

The argument `data` is a string of 8-bit characters that are transferred, without any translation, to the GPIB. The argument `count` is a numeric string preceded by a number sign (`#`). `count` specifies the number of GPIB data bytes to write, and can range from 1 to 4294967295. `count` must not include the <LF> or <CR> characters, which are used to terminate the programming message or string of GPIB data bytes.

If you call `wrt` without `count`, the GPIB-RS232/485/422 recognizes the end of the data string when it sees a <CR> or <LF>. Consequently, `count` is required only if the data string contains a <CR> or <LF> character.

The argument `alist` is a list of addresses separated by commas or spaces. Addresses specify the GPIB addresses of the Listeners.

Do not terminate the `wrt` programming message with the <CR> character alone. Use

either <LF> or <CR><LF> to terminate the programming message. If you use <CR> alone, and the first character of the data string is <LF>, the GPIB-RS232/485/422 discards the <LF> as the second character of a <CR><LF> termination.

The GPIB-RS232/485/422 writes data to the GPIB until one of the following events occurs:

- The GPIB-RS232/485/422 successfully transfers all data.
- The I/O time limit is exceeded. The EABO error is recorded.
- The GPIB-RS232/485/422 receives a Device Clear. The EABO error is recorded.
- There is no listening device on the bus receiving the data. The ENOL error is recorded.
- The `alist` argument is specified, and the requested GPIB addressing bytes cannot be sent. The EBUS error is recorded.
- A serial port error occurs and is not ignored (refer to `spign`).

After `wrt` terminates, the GPIB-RS232/485/422 records the number of data bytes it actually sent. If one of the errors described above occurs, the count might be less than expected.

If an error occurs and the GPIB-RS232/485/422 cannot transmit the entire data string, the GPIB-RS232/485/422 reads in and discards the remaining data string bytes.

If you call `wrt` with the `alist` argument, the GPIB-RS232/485/422 must be CIC to perform the addressing. If this is the first function you call that requires GPIB Controller capability, and you have not disabled System Controller capability with `rsc`, the GPIB-RS232/485/422 sends Interface Clear (IFC*) to make itself CIC.

If you call `wrt` with the `alist` argument and you previously passed control to some other GPIB device, control must be passed back to you or you must send IFC* to make yourself CIC before making this call. Otherwise, the ECIC error is recorded.

When performing the addressing for a specified `alist`, the GPIB-RS232/485/422 sends out its own talk address as well as the listen addresses of the specified devices. It then places itself in Standby Controller state with ATN* unasserted and remains there after the write operation is complete.

If the `alist` argument is not specified, the GPIB-RS232/485/422 assumes the

Controller already has addressed it. If the GPIB-RS232/485/422 is the Controller and did not address itself to talk before calling `wrt`, the EADR error is recorded and no data bytes are transferred.

See Also

`eos`, `eot`, `spign`, and `tmo`.

Example 1

Write 10 bytes to a device at primary address 9 and secondary address 97.

```
char  buffer [] = "wrt #10 9+97\n0123456789\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

Example 2

Write the data bytes `ABCDE` to the device at address 2.

```
char  buffer [] = "wrt 2\nABCDE\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

xon

Change Serial Port XON/XOFF Protocol

Type

Serial port function

Syntax

xon [booltx] [,boolrx]<CR>

Purpose

You can use `xon` at the beginning of your program to configure the GPIB-RS232/485/422 to communicate over the serial port using the same XON/XOFF protocol as your serial device.

Remarks

The argument `booltx` specifies whether to enable the XON/XOFF protocol when sending data out on the serial port. If the argument `booltx` is `1`, the GPIB-RS232/485/422 monitors its serial receive buffer for XON/XOFF characters as it sends data over the serial port. If it receives the XOFF character (decimal 19 or <Ctrl-S>), it immediately stops sending data. When it receives the XON character (decimal 17 or <Ctrl-Q>), it begins sending data again. To send a data string that might contain a <Ctrl-S> or <Ctrl-Q>, you must disable `booltx`.

The argument `boolrx` specifies whether to enable the XON/XOFF protocol when receiving data over the serial port. If the argument `boolrx` is 1, and the GPIB-RS232/485/422 is receiving data over the serial port, it sends XOFF over the serial port when its serial receive buffer is almost full. This tells the sender to stop sending data. When the GPIB-232/485CT-A serial port receive buffer is ready to receive more bytes, the GPIB-RS232/485/422 sends XON over the serial port. This tells the sender to begin sending data again.



Note Enabling either `booltx` or `boolrx` enables the XON/XOFF protocol for both sending and receiving the data to/from the serial port. This behavior is different from the GPIB-232-CT-A. Refer to Differences between the GPIB-232/485CT/CV-A and GPIB-RS232/485/422 for more information about this and other differences.

You should use XON/XOFF if your serial device does not recognize the hardware handshake protocol and you are transferring large amounts of data at high speeds. Without handshaking, there is danger of overflowing the GPIB-RS232/485/422 and serial device serial port receive buffers.

The flow control setting selected using the NI GPIB-Serial Converter Wizard determines the power-on default of XON/XOFF for both sending and receiving. For more information, refer to Configuration of S Mode and Hardware Handshaking.

If you call `xon` without an argument, the GPIB-RS232/485/422 returns the current handshake setting: 1 if the protocol is enabled or 0 if protocol is disabled.

The assignment made by `xon` remains in effect until you call `xon` again, call `onl`, or turn off the GPIB-RS232/485/422.

Example 1

Enable the GPIB-RS232/485/422 XON/XOFF protocol for TX and RX.

```
char buffer [] = "xon 1,1\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

Example 2

Disable the GPIB-RS232/485/422 XON/XOFF protocol for TX and RX.

```
char  buffer [] = "xon 0,0\r";
viWrite(instr, buffer, strlen(buffer), &retCount);
```

Programming in G Mode

The following topics describe how to program the GPIB-RS232/485/422 in G Mode. They describe status and error handling information, programming considerations, programming messages, function arguments, addressing in G Mode, GPIB read and write termination methods, serial port transmissions, and the G Mode functions.

The topics also explain how to communicate with your serial device through the GPIB-RS232/485/422.

Status Information and Error Handling Characteristics

Programming Considerations

Programming Messages

Programming Message Format

Programming Message Example 1

Programming Message Example 2

How Messages are Processed

Function Arguments

Abbreviations for Arguments

Addressing the GPIB-RS232/485/422 and Serial Device

Address of the GPIB-RS232/485/422

Address of the Serial Device

Addressing the GPIB-RS232/485/422 and Serial Device as Listeners

Addressing the GPIB-RS232/485/422 and Serial Device as Talkers

GPIB Read and Write Termination Methods (END and EOS)

Serial Port Transmission

Operation of the GPIB-RS232/485/422 as a GPIB Device

Serial Poll Responses

Service Request Conditions

Parallel Polls

Take Control (TCT)

Group Execute Trigger (GET)

Go To Local (GTL)

Device Clear

Function Names

G Mode Default Settings and Related Functions

List of G Mode Functions by Group

GPIB Functions

Serial Port Functions

General-Use Functions

Alphabetical List of G Mode Functions

Status Information and Error Handling Characteristics

The function descriptions in G Mode Functions explain that the GPIB-RS232/485/422 records specific status and error information. This means it stores the information in its memory, so the information is available when you request it.

The function descriptions also explain that the GPIB-RS232/485/422 returns certain information to you. This means the GPIB-RS232/485/422 sends information to you over the GPIB.

The GPIB-RS232/485/422 continuously monitors the serial port for transmission errors. If it encounters an error in the serial data, the GPIB-RS232/485/422 records the error. You can program the GPIB-RS232/485/422 to ignore serial port errors using the `spign` function.

Programming Considerations

The program examples for each function are written in ANSI C, using NI-VISA. Although the examples in this manual are written in C, you can program the GPIB-RS232/485/422 using any programming language with access to a GPIB port.

The following VISA function addresses the GPIB-RS232/485/422 to listen and sends it the programming message `spset`, followed by a carriage return:

```
viWrite (GPIB232, "spset\r", 6)
```

The code above assumes you properly initialized VISA resource and used `viOpen()` to obtain the GPIB232 handle. For more details, refer to GPIB Initialization Code Using VISA.

If you are not using NI-VISA, be sure your program properly addresses the GPIB-RS232/485/422 and serial device when writing to and reading from them.

In the function syntax descriptions, arguments shown in square brackets (`[]`) are optional. Do not enter the brackets as part of your argument.

You can use function name abbreviations, which include only as many characters as necessary to distinguish them from other functions. The abbreviated forms are indicated by bold text in the syntax description of each function.

Programming Messages

You can program the GPIB-RS232/485/422 by sending it programming messages, which are ASCII strings, by way of its GPIB port.

Programming Message Format

Each programming message is terminated with a carriage return (<CR>), a linefeed (<LF>), or a carriage return followed by a linefeed (<CR><LF>). This is denoted by <CR> in the syntax portions of the function descriptions and by `\r` in the NI-VISA examples.

You can enter programming messages in any combination of uppercase and lowercase letters.

Programming Message Example 1

The following line of code is an example of a programming message in NI-VISA:

```
viWrite (GPIB232, "EOS x,10\r", 9);
```

The programming message contains the function name, `eos`. `x` and `10` are the arguments, and `\r` is the terminating carriage return. This programming message tells the GPIB-RS232/485/422 to assert the EOI* line when it sends the end-of-string character linefeed. The `viWrite` is an NI-VISA function call that outputs the string `eos x,10\r` to the GPIB-RS232/485/422.

Programming Message Example 2

To send more than one programming message with one GPIB write operation, embed a <CR> or <LF> in the data string you send. For example, to send the two programming messages `change serial port parameters` and `change serial port XON/XOFF protocol`, you could use either of the two following sequences:

- You can send two separate strings to the GPIB in two separate GPIB writes, as in the following example:

  ```
  viWrite(GPIB232, "SPSET 1200,n,8\r", 15);
  viWrite(GPIB232, "XON 1,1\r", 8);
  ```

- You could also put both messages in one string and send it to the GPIB-RS232/485/422 in one GPIB write, as in the following example:

  ```
  viWrite(GPIB232, "SPSET 1200,n,8\rXON 1,1\r", 23);
  ```

How Messages Are Processed

The GPIB-RS232/485/422 processes each programming message on a line-by-line basis. When the GPIB-RS232/485/422 receives a message, it buffers the message, interprets the function name and arguments, and then executes the message.

Function Arguments

When you type in a function, separate the first argument from the function name with at least one space. Separate each additional argument with at least one space or comma.

In the syntax portions of the function descriptions in G Mode Functions, the square brackets (`[]`) that enclose some arguments indicate that those arguments are optional. Do not enter the brackets as part of your arguments.

Abbreviations for Arguments

The term `bool` is an abbreviation used for an argument in the function descriptions. The values for `bool` are `1` = true, on, or enable, and `0` = false, off, or disable.

Addressing the GPIB-RS232/485/422 and Serial Device

The GPIB-RS232/485/422 uses dual addressing to determine what type of GPIB data it is processing. With dual addressing, the GPIB-RS232/485/422 recognizes two different GPIB addresses. The first address is the GPIB-RS232/485/422 address. Data sent by a GPIB controller while the GPIB-RS232/485/422 is addressed is interpreted as programming messages. The data the GPIB-RS232/485/422 sends is referred to as status information. The second address is the serial device address. When the serial device is addressed, the data it sends and receives is referred to as serial data.

Address of the GPIB-RS232/485/422

The GPIB-RS232/485/422 address is the primary and optionally secondary address you set in Using the NI GPIB-Serial Converter Wizard to Configure the GPIB-RS232/485/422, choosing G Mode as the run mode. The factory power-on default of this address is primary address 5 and secondary address disabled.

Address of the Serial Device

The serial device address is the primary and optionally secondary address you set in Using the NI GPIB-Serial Converter Wizard to Configure the GPIB-RS232/485/422, choosing G Mode as the run mode. The factory power-on default of this address is primary address 6 and secondary address disabled.

Addressing the GPIB-RS232/485/422 and Serial Device as Listeners

When the GPIB-RS232/485/422 receives its own listen address, it examines the data

received over the GPIB, treats it as a programming message, and takes actions based on that data.

When the GPIB-RS232/485/422 receives the serial device listen address, it forwards the data received over the GPIB to the serial port without examining or modifying the data.

For example, you might have a serial printer connected to the GPIB-RS232/485/422, and you want to send a data file from your computer over the GPIB to the printer. Ordinarily, when the printer buffer is full, the printer sends the XOFF character; when the printer is ready to receive more characters, it sends the XON character. So, before you send your file to the printer, you must tell the GPIB-RS232/485/422 to watch for XON/XOFF characters from the printer.

Follow these steps to send a data file from your computer to a serial printer:

1. Address the GPIB-RS232/485/422 to listen by sending its listen address.
2. Send the programming message `xon ,1` to the GPIB-RS232/485/422. The GPIB-RS232/485/422 interprets this programming message and acts on it without sending any data on to the serial device.
3. Address the serial device to listen by sending the serial device listen address to the GPIB-RS232/485/422.
4. Send your data file over the GPIB. The GPIB-RS232/485/422 sends the data to the printer without examining it for meaning.

   The **LSTN** LED on the GPIB-RS232/485/422 is lit when either the GPIB-RS232/485/422 or serial device is addressed to listen.

Addressing the GPIB-RS232/485/422 and Serial Device as Talkers

When the GPIB-RS232/485/422 receives its own talk address, it sends out status information.

When the GPIB-RS232/485/422 receives the serial device talk address, it sends data out to the GPIB that it has received from the serial device.

For example, you might have your serial device programmed to perform some calculations, and you want it to return the data to you. Follow these steps to retrieve the data:

1. Address the serial device to talk by sending the serial device talk address to the GPIB-RS232/485/422. If the GPIB-RS232/485/422 receives the serial device talk address, but has no data in its serial port receive buffer to send, it waits for data from the serial device to fill the request.
2. Perform a GPIB read of 100 bytes. The GPIB-RS232/485/422 retrieves 100 bytes from its serial port receive buffer and sends them to you.
3. Find out if the serial device has sent more bytes to the GPIB-RS232/485/422 by asking the GPIB-RS232/485/422 to send you status information.
    1. Send the GPIB-RS232/485/422 its listen address.
    2. Send the programming message `stat n` to the GPIB-RS232/485/422.
    3. Send the GPIB-RS232/485/422 its talk address.
    4. Perform a GPIB read of 20 bytes. The GPIB-RS232/485/422 sends you its status information, terminated by the GPIB END message. If the GPIB-RS232/485/422 receives its talk address but has nothing to send, it does not respond to GPIB reads. In this example, it can occur if you forget to execute steps 3a and 3b before 3c and 3d. This results in timeout termination of GPIB read on the host.

    The status information returned to you contains the number of bytes remaining in the serial port receive buffer. This information helps you decide how much data to continue to collect from the serial device.

The **TALK** LED on the GPIB-RS232/485/422 is lit when either the GPIB-RS232/485/422 or serial device is addressed to talk.

GPIB Read and Write Termination Methods (END and EOS)

The IEEE 488 specification defines two ways that GPIB Talkers and Listeners can identify the last byte of data messages: END and EOS. The two methods permit a Talker to send data messages of any length without the Listener(s) knowing the number of transmission bytes in advance. You can use END and EOS individually or in combination, but you must configure the Listener to detect the end of a transmission.

END message	The Talker asserts the EOI* (End or Identify) signal while the last data byte is being transmitted. The Listener stops reading when it detects a data byte accompanied by EOI*, regardless of the byte value.
EOS character	The Talker transmits an EOS (end-of-string) character at the end of its data string. The Listener stops receiving data when it detects the EOS character. Either a 7-bit ASCII character or a full 8-bit binary byte can be used.

When the GPIB-RS232/485/422 receives its own talk or listen address, no EOS Modes are in effect. When talking, the GPIB-RS232/485/422 asserts EOI* with the last byte of its response. When the GPIB-RS232/485/422 receives the serial device talk address, the EOS Modes in effect are those you select using the `eos` function.

Serial Port Transmission

The GPIB-RS232/485/422 checks the data received from the serial device for errors while it buffers data. If a serial port error occurs, the GPIB-RS232/485/422 records the appropriate error code. There are two ways to determine if a serial port error has occurred:

- Use the `stat` function to request the GPIB-RS232/485/422 status information. After the serial error code has been reported, it is cleared automatically, and no further action to the GPIB-RS232/485/422 is necessary.
- Serial poll the GPIB-RS232/485/422 and check the serial poll response byte to see if its SERR bit is set. Refer to Serial Poll Responses for more information.

You can program the GPIB-RS232/485/422 to ignore serial port errors using the `spign` function.

Operation of the GPIB-RS232/485/422 as a GPIB Device

In G Mode, the GPIB-RS232/485/422 operates like any other GPIB device and, as such, is configured to respond in certain ways to GPIB commands.

Serial Poll Responses

When serial polled, the GPIB-RS232/485/422 returns status information to the GPIB Controller through the serial poll response byte. The GPIB-RS232/485/422 maintains this status byte throughout operation, regardless of the `srqen` configuration.

Note Be sure to serial poll the GPIB-RS232/485/422 address and not the serial device address to obtain the serial poll response byte. For more information about addressing, refer to Addressing the GPIB-RS232/485/422 and Serial Device. Unlike the GPIB-232/485/CT/CV-A, which responded to serial polls on either address, the GPIB-RS232/485/422 responds only to

	serial polls on the GPIB-RS232/485/422 address. For more information about this and other differences, refer to Differences Between the GPIB-232/485CT/CV-A and GPIB-RS232/485/422.

The following table lists the meaning of each bit in the serial poll response byte.

Serial Poll Response Byte

Bit	Mnemonic	Meaning
0	—	Not Used
1	BF	Serial port receive buffer is full
2	GERR	GPIB error—refer to stat
3	SERR	Serial error—refer to stat
4	BNE	Serial port receive buffer not empty
5	EOS	EOS character received
6	RQS	Request service (SRQ* asserted)
7	—	Not Used

Service Request Conditions

You can program the GPIB-RS232/485/422 to assert Service Request (SRQ*) in a variety of cases. After you power on, the GPIB-RS232/485/422 defaults to never asserting Service Request. Using the `srqen` function, you can program the GPIB-RS232/485/422 to assert SRQ* under any of the following circumstances:

- When a GPIB error occurs—that is, EARG, ECMD, or ECAP as reported by `stat`.
- When a serial port error occurs—that is, EPAR, EORN, EOFL, or EFRM as reported by `stat`.
- When the serial port receive buffer is full, any byte is received from the attached serial device, or the EOS byte is received from the attached serial device.

Parallel Polls

Parallel poll functionality is not supported at this time. You should not perform a parallel poll and expect a valid parallel poll response from the GPIB-RS232/485/422 in

G Mode.

Note This functionality is different from the GPIB-232/485CT/CV-A. Refer to Differences Between the GPIB-232/485CT/CV-A and GPIB-RS232/485/422 for more information about this and other differences.

Take Control (TCT)

In G Mode, the GPIB-RS232/485/422 can act only as a GPIB Talker or Listener. Do not pass control to the GPIB-RS232/485/422 in G Mode. If you do, the GPIB circuitry in the GPIB-RS232/485/422 accepts control and immediately asserts the ATN* signal, a system error condition you should avoid.

Group Execute Trigger (GET)

This command has no effect on the GPIB-RS232/485/422.

Go To Local (GTL)

This command has no effect on the GPIB-RS232/485/422.

Device Clear

When the GPIB-RS232/485/422 receives the universal Device Clear (DCL) command, or its listen address and the Selected Device Clear (SDC) command, it clears both its status buffer and serial port receive buffer. It also resets the GPIB serial poll response byte to 0 and unasserts SRQ*.

Function Names

The function names have been selected to indicate each function's purpose, thereby making your programs easy to understand. However, if you want to reduce some overhead in your program and do not mind giving up these advantages, you can use only as much of the function name as is necessary to distinguish it from other functions. This abbreviated form of the function name is shown in bold text in the function tables and in the syntax portions of the function descriptions.

G Mode Default Settings and Related Functions

The following tables list power-on characteristics of the GPIB-RS232/485/422 and the functions you can use to change those characteristics.

G Mode Serial Port Characteristics

Characteristic	Power-On Value	Related Function
Echo bytes to serial port	no	`echo`
Enable serial port communication	yes	`onl`
Baud rate	9600	`spset`
Parity	none	`spset`
Data bits	8	`spset`
Stop bits	1	`spset`
Hardware handshaking	on	`xon`
Send XON/XOFF	no	`xon`
Recognize XON/XOFF	no	`xon`
Report serial errors	no	`spign`

G Mode GPIB Characteristics

Characteristic	Power-On Value	Related Function
End-of-string modes	none	`eos`
Allow GPIB-RS232/485/422 to assert SRQ	no	`srqen`

List of G Mode Functions by Group

The GPIB-RS232/485/422 functions are divided into three groups: GPIB functions, serial port functions, and general-use functions. For more information about the G Mode functions, refer to Alphabetical List of G Mode Functions or G Mode Functions. The following topics list programming messages sent to the GPIB-RS232/485/422 from a GPIB Talker.

GPIB Functions

Serial Port Functions

General-Use Functions

GPIB Functions

GPIB functions manage the GPIB-RS232/485/422 GPIB port.

- eos
- **srq**en

Serial Port Functions

Serial port functions initialize and manage the GPIB-RS232/485/422 serial port.

- **ec**ho
- **sps**et
- **x**on
- **spi**gn

General-Use Functions

The general-use functions are for general operations that the GPIB functions or serial port functions do not provide.

- **id**
- **o**nl
- **st**at

Alphabetical List of G Mode Functions

The following table lists the G Mode functions in alphabetical order.

Function	Purpose

echo on/off	Echo characters received from serial port
eos modes, eoschar	Change or disable GPIB end-of-string termination mode
id	Identify system
onl on/off	Place the GPIB-RS232/485/422 online/offline
spign on/off	Ignore serial port errors
spset modes	Change serial port parameters
srqen mask	Set conditions for asserting SRQ*
stat options	Return GPIB-RS232/485/422 status
xon modes	Change serial port XON/XOFF protocol

G Mode Functions

The G Mode function descriptions describe the G Mode functions you can use to program the GPIB-RS232/485/422.

For general information about using G Mode functions, refer to Programming in G Mode.

For more information, refer to General Programming Steps for Communicating with the GPIB-RS232/485/422 Over GPIB.

echo

Echo Characters Received from Serial Port

Type

Serial port function

Syntax

echo [bool]<CR>

Purpose

You can use `echo` when a terminal is connected to the GPIB-RS232/485/422 and you want everything you type to display on the screen.

Remarks

If the argument `bool` is `1`, characters received from the serial port are echoed back to the serial port. If the argument `bool` is `0`, characters are not echoed. By default, echoing is disabled.

If you call `echo` without an argument, the GPIB-RS232/485/422 returns the current setting.

In a debugging environment, the success of your communication with the serial device might be unclear. In this case, you could connect a terminal to the GPIB-RS232/485/422 instead of connecting the serial device. Then the data that the GPIB-RS232/485/422 would send to the serial device is displayed on the terminal screen. Also, you can type characters on the terminal to send to the GPIB-RS232/485/422, just as your serial device would.

The assignment made by `echo` remains in effect until you call `echo` again, call `onl`, or turn off the GPIB-RS232/485/422.

Example 1

```
// Enable character echoing.
char echoCommand [] = "echo 1\r";
viWrite (GPIB232, echoCommand, strlen(echoCommand), &retCount);
```

Example 2

```
// Disable character echoing.
char echoCommand [] = "echo 0\r";
viWrite (GPIB232, echoCommand, strlen(echoCommand), &retCount);
```

Example 3

```
// What is the current echo status?
char echoQuery [] = "echo\r";
char buffer [10];
viWrite (GPIB232, echoQuery, strlen(echoQuery), &retCount);
viRead  (GPIB232, buffer, 10, &retCount);
// buffer contains
// 0<CR><LF>
// (character echo is
// disabled).
```

eos

Change/Disable GPIB EOS Termination Mode

Type

GPIB function

Syntax

eos `[X[B]eoschar]`<CR>

or

eos `D`<CR>

Purpose

You can use `eos` to enable the GPIB-RS232/485/422 to add the GPIB END message to the data string sent by the serial device when the string contains the specified end-of-string character.

Remarks

`eos` applies only when the GPIB-RS232/485/422 has received the serial device talk address and is sending serial data to the GPIB. It does not apply when the GPIB-RS232/

485/422 reads serial data or programming messages from the GPIB. The arguments `X`, `B`, and `D` specify GPIB data transfer termination methods.

The argument `eoschar` is a numeric string representing a single ASCII character that is to be the EOS byte. The arguments `X` and `B` enable the corresponding EOS Mode. The argument `D` disables all EOS Modes.

The three termination methods are shown in the following table.

Data Transfer Termination Methods in G Mode

Description	Letter
`XEOS`—Set EOI* with EOS when sending data from serial device	`X`
`BIN`—compare all 8 bits of EOS byte rather than low 7 bits	`B`
`DISABLE`—disable all EOS Modes	`D`

If methods `X` or `X` and `B` are chosen, the GPIB-RS232/485/422 automatically sends the END message along with `eoschar` when performing GPIB writes of serial data. That is, when the GPIB-RS232/485/422 receives `eoschar` over the serial port and sends it on to the GPIB, it also asserts EOI* along with that byte. When `X` alone is chosen, END is sent with the EOS byte when the low seven bits of that byte match the low seven bits of eoschar. When `X` and `B` are chosen, a full 8-bit comparison is used.

If `B` is the only mode chosen, the EARG error is posted.

If `D` is chosen, all EOS Modes are disabled.

By default, all EOS Modes are disabled.

If you call `eos` without an argument, the GPIB-RS232/485/422 returns the current `eos` settings.

The assignment this function makes remains in effect until you call `eos` again, call `on l`, or turn off the GPIB-RS232/485/422.

See Also

`srqen` and GPIB Read and Write Termination Methods (END and EOS).

Example 1

```
// Send EOI with <CR>. Compare 7 bits.
char eosCommand [] = "eos X 13\r";
viWrite (GPIB232, eosCommand, strlen(eosCommand), &retCount);
```

Example 2

```
// What are the current EOS settings?
char eosQuery [] = "eos\r";
char buffer [10];
viWrite (GPIB232, eosQuery, strlen(eosQuery), &retCount);
viRead  (GPIB232, buffer, 10, &retCount);
// buffer contains
// X,13<CR><LF>.
```

id

Identify System

Type

General-use function

Syntax

id<CR>

Purpose

You can use `id` to know the revision level of the software in your GPIB-RS232/485/422.

Remarks

The identification is returned in three strings. The first string identifies the company product model. The second string is a copyright notice. The third string identifies the GPIB-RS232/485/422 firmware version.

The following example shows the current identification string at the time of this writing. The general format is as shown; however, version-specific information such as revision levels and copyright dates change as needed.



Note There are format differences between the GPIB-RS232/485/422 and GPIB-232/485CT/CV-A regarding the id string returned. For more information about this and other behavioral differences, refer to Differences Between the GPIB-232/485CT/CV-A and GPIB-RS232/485/422.

Example 1

```
// Get GPIB-RS232 system identification
char idQuery [] = "id\r";
char buffer [100];
memset(buffer,0,100);
viWrite (GPIB232, idQuery, strlen(idQuery), &retCount);
viRead  (GPIB232, buffer, 100, &retCount);
```

onl

Place the GPIB-RS232/485/422 Online/Offline

Type

General-use function

Syntax

onl [bool]<CR>

Purpose

You can use `onl` to disable communication between the GPIB-RS232/485/422 and serial port or reset the GPIB-RS232/485/422 characteristics to their default values.

Remarks

If the argument `bool` is `1`, the GPIB-RS232/485/422 places itself online. If the argument `bool` is `0`, it places itself offline. By default, the GPIB-RS232/485/422 powers up online.

If you call `onl` without an argument, the GPIB-RS232/485/422 returns its current state, which is `0` if the GPIB-RS232/485/422 is offline and `1` if the it is online.

Placing the GPIB-RS232/485/422 offline is like disconnecting its serial cable from the serial device. When placed offline, no data is sent out the GPIB-RS232/485/422 serial port, and data coming in to the GPIB-RS232/485/422 serial port is not saved until `onl` is called with `bool` = `1`.

Placing the GPIB-RS232/485/422 online enables data to be sent and received over its serial port. Placing the GPIB-RS232/485/422 online also restores all its settings to their power-on default values and discards any serial and GPIB data buffered inside the GPIB-RS232/485/422.

See Also

G Mode Default Settings and Related Functions.

Example 1

```
// Put the GPIB-RS232/485/422 online and restore its power-on settings.
char onlCommand [ ] = "onl 1\r";
viWrite (GPIB232, onlCommand, strlen(onlCommand), &retCount);
```

Example 2

```
// Put the GPIB-RS232/485/422 offline.
```

```
char onlCommand [ ] = "onl 0\r";
viWrite (GPIB232, onlCommand, strlen(onlCommand), &retCount);
```

spign

Ignore Serial Port Errors

Type

Serial port function

Syntax

spi`gn [bool]`<CR>

Purpose

You can use `spign` at the beginning of your program to change the effect serial port errors have on the storage of a character received with a serial error. This function tells the GPIB-RS232/485/422 to ignore or not to ignore the occurrence of serial port errors. By default, the GPIB-RS232/485/422 ignores serial port errors.

Remarks

If the argument `bool` is `0`, the GPIB-RS232/485/422 does not ignore serial port errors. In this case, the GPIB-RS232/485/422 does not store characters that contain serial errors. Also, the error is indicated in the serial error code the `stat` function returns.

The serial port errors include parity, overrun, framing, and overflow errors. Status and Error Message Information includes a list of serial port errors.

If the argument `bool` is `1`, the GPIB-RS232/485/422 ignores serial port errors. Bytes that arrive with serial port errors are stored in the buffer and are later sent out of the GPIB port when the serial device is talk addressed. The `stat` function reports no serial error code.

If you call `spign` without an argument, the GPIB-RS232/485/422 returns the current setting.

The assignment `spign` makes remains in effect until you call `spign` again, call `onl`, or turn off the GPIB-RS232/485/422.

Example 1

```
// Do not ignore serial port errors.
char spignCommand [] = "spign 0\r";
viWrite (GPIB232, spignCommand, strlen(spignCommand), &retCount);
```

Example 2

```
// Ignore serial port errors.
char spignCommand [] = "spign 1\r";
viWrite (GPIB232, spignCommand, strlen(spignCommand), &retCount);
```

spset

Change Serial Port Parameters

Type

Serial port function

Syntax

sps`et [baud] [parity] [databits] [stopbits]`<CR>

Purpose

You can use `spset` at the beginning of your program to set the GPIB-RS232/485/422 serial port characteristics (baud rate, parity, data bits, and stop bits) to match those your serial device requires.

Remarks

The argument `baud` is a numeric string specifying the baud rate (300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 76800, or 115200). The argument `parity` is a character specifying the parity (`e` for even, `o` for odd, `n` for none). The argument `databits` is a character specifying the number of data bits (7 or 8). The argument `stopbits` is a character specifying the number of stop bits (1 or 2). If you call `spset` without an argument, the GPIB-RS232/485/422 returns its current serial port configuration.

Until you call `spset`, the following characteristics are in effect: 9600 is the baud rate, parity is disabled, 8 is the number of data bits, and 1 is the number of stop bits.

If you must reconfigure the GPIB-RS232/485/422 serial port, wait until communication with the serial port is not taking place. Remember that calling `spset` with one or more arguments also discards any serial data buffered inside the GPIB-RS232/485/422.

The assignment `spset` makes remains in effect until you call `spset` again, call `onl`, or turn off the GPIB-RS232/485/422.

Example 1

```
// Set up the serial port of GPIB-RS232/485/422 to
// keep its current baud rate, current parity,
// and to use 7 data bits and 2 stop bits.
char spsetCommand [] = "spset 7 2\r";
viWrite (GPIB232, spsetCommand, strlen(spsetCommand), &retCount);
```

Example 2

```
// What are the current GPIB-RS232/485/422 serial port settings?
char spsetQuery [] = "spset\r";
char buffer [30];

viWrite (GPIB232, spsetQuery, strlen(spsetQuery), &retCount);
viRead  (GPIB232, buffer, 30, &retCount);
// buffer will contain 9600 N 7 2<CR><LF>
// (9600 baud, no parity, 7 data bits,
// 2 stop bits).
```

Example 3

```
// Set the GPIB-RS232/485/422 serial port to 115200
// baud, no parity, 8 data bits, and 1 stop bit.
char spsetCommand [] = "spset 115200 n 8 1\r";
viWrite (GPIB232, spsetCommand, strlen(spsetCommand), &retCount);
```

srqen

Enable/Disable Setting of SRQ*

Type

GPIB function

Syntax

srq`en [mask]`<CR>

Purpose

You can use `srqen` when you want to allow the GPIB-RS232/485/422 to assert SRQ* under the conditions described in Service Request Conditions.

Remarks

When the argument `mask` is `0`, the GPIB-RS232/485/422 never asserts SRQ*. When the argument `mask` is greater than 0, the GPIB-RS232/485/422 asserts SRQ* under the conditions each mask bit represents. The following table describes the mask bits.

SRQ Mask Bits in G Mode

Bit	Hex Value	Decimal Value	Mnemonic	Description
0	1	1	—	Not used
1	2	2	BF	Serial port receive buffer full
2	4	4	GERR	GPIB error; refer to `stat`

3	8	8	SERR	Serial error; refer to stat
4	10	16	BNE	Serial port receive buffer not empty
5	20	32	EOS	EOS character received
6	40	64	—	Not used
7	80	128	—	Not used

To determine the mask value you want, add the hex or decimal values of the conditions on which you want SRQ* to be asserted. For example, if you want SRQ* asserted on GPIB errors and serial port errors, call `srqen` with a mask of `12` (`4` for GERR and `8` for SERR).

The `srqen` power-on default is disabled—that is, SRQ* is never asserted.

If you call `srqen` without an argument, the GPIB-RS232/485/422 returns a decimal string indicating the the current setting decimal value.

The assignment `srqen` makes remains in effect until you call `srqen` again, call `onl`, or turn off the GPIB-RS232/485/422.

See Also

eos, spign, and stat.

Example 1

```
// Never assert SRQ*
char srqenCommand [] = "srqen 0\r";
viWrite (GPIB232, srqenCommand, strlen(srqenCommand), &retCount);
```

Example 2

```
// Assert SRQ* when a GPIB error occurs.
char srqenCommand [] = "srqen 4\r";
viWrite (GPIB232, srqenCommand, strlen (srqenCommand), &retCount);
```

stat

Return GPIB-RS232/485/422 Status

Type

General-use function

Syntax

st`at [[c] n]`<CR>

or

st`at [c] s`<CR>

or

st`at [c] n s`<CR>

Purpose

You can use `stat` to obtain the GPIB-RS232/485/422 status to see if certain conditions are currently present. `stat` is used most often to see if the previous operation resulted in an error.

Remarks

The GPIB-RS232/485/422 responds with status information in a form depending on the mode or combination of modes you choose. `n` indicates that the status information is returned as numeric strings. `s` indicates that the status information is returned in symbolic format—that is, as mnemonic strings. Status bits are separated by a comma. Normally, you use `s` only when you are debugging your code and want to print the mnemonic for each piece of status information. `c` specifies that the status is returned after each programming message, eliminating the need to call `stat` after each programming message.

You would probably use `s`, or symbolic format, only when you are debugging your code and you want to print the mnemonic for each piece of status information.

If you call `stat` without an argument, continuous status reporting is disabled.

The status information returned by the GPIB-RS232/485/422 contains four pieces of information: the GPIB-RS232/485/422 status, a GPIB-error code, a serial-error code, and a count. The GPIB-RS232/485/422 returns a <CR><LF> following each piece of the response and asserts EOI* with the final <LF> that comes after count.

Status represents a combination of GPIB-RS232/485/422 conditions. Inside the GPIB-RS232/485/422, status is stored as a 16-bit integer. Each bit in the integer represents a single condition. A bit value of 1 indicates the corresponding condition is in effect. A bit value of 0 indicates the condition is not in effect. Because more than one GPIB-RS232/485/422 condition can exist at one time, more than one bit can be set in status. The highest order bit of status, also called the sign bit, is set when the GPIB-RS232/485/422 detects either a GPIB error or a serial port error. When status is negative, an error condition exists.

The following table lists the values and descriptions of GPIB status conditions the `stat` function might return.

G Mode Conditions `stat` Returns

Numeric Value (n)	Symbolic Value (s)	Description	Bit
–32768	ERR	Error detected	15
16384	—	Reserved	14
8192	—	Reserved	13
4096	—	Reserved	12
2048	—	Reserved	11
1024	—	Reserved	10
512	—	Reserved	9
256	CMPL	Operation completed	8
128	—	Reserved	7
64	—	Reserved	6
32	—	Reserved	5
16	—	Reserved	4

8	—	Reserved	3
4	—	Reserved	2
2	—	Reserved	1
1	—	Reserved	0

The GPIB error code represents a single GPIB error condition present. The serial error code represents a single serial error condition present. `count` is the number of bytes currently contained in the GPIB-RS232/485/422 serial port receive buffer.

The following table lists the values and descriptions of GPIB error conditions the `stat` function might return.

G Mode GPIB Error Conditions `stat` Returns

Numeric Value (n)	Symbolic Value (s)	Description
0	NGER	No GPIB error condition to report
1	—	Reserved
2	—	Reserved
3	—	Reserved
4	EARG	Invalid argument or arguments
5	—	Reserved
6	—	Reserved
7-10	—	Reserved
11	ECAP	No capability for operation
12-13	—	Reserved
14	—	Reserved
15-16	—	Reserved
17	ECMD	Unrecognized command

The following table lists the serial port errors the `stat` function might return.

Serial Port Error Conditions `stat` Returns

Numeric Value (n)	Symbolic Value (s)	Description
0	NSER	No serial port error condition to report
1	EPAR	Serial port parity error
2	EORN	Serial port overrun error
3	EOFL	Serial port receive buffer overflow
4	EFRM	Serial port framing error

Status and Error Message Information includes a detailed description of the conditions under which each bit in status is set or cleared.

The GPIB-RS232/485/422 updates the status and count variables at the end of each programming message. The serial and GPIB error variables are updated whenever a new error occurs and are cleared automatically when status is reported.

When you want to begin continuous status reporting, send the `stat c s<CR>`, `stat c n<CR>`, or `stat c n s<CR>` programming message. When you call `stat` with both `n` and `s` modes specified, the numeric status is always returned first.

The continuous status setting remains in effect until you call `stat` again, call `onl`, or turn off the GPIB-RS232/485/422.

Example 1

```
char statCommand [] = "stat n\r";
char statusString [11];
char gpibERR [11];
char serialERR [11];
char count [11];
int statusValue;

memset(statusString,0,11);
memset(gpibERR,0,11);
memset(serialERR,0,11);
memset(count,0,11);

//Enable EOS=<LF> in VISA so that viRead terminates on linefeed
```

```
viSetAttribute (GPIB232, VI_ATTR_TERMCHAR, 10);
viSetAttribute (GPIB232, VI_ATTR_TERMCHAR_EN, VI_TRUE);


// Tell GPIB-RS232/485/422 to send numeric
// status.
viWrite (GPIB232, statCommand, strlen(statCommand), &retCount);


// Read up to 10 bytes of each piece of status.
// The GPIB-RS232/485/422 returns 4 pieces of
// status. We are set up to terminate
// viRead on linefeed, which is what
// terminates each piece of status.

// Now read the status from the
// GPIB-RS232/485/422.
viRead (GPIB232, statusString, 10, &retCount);
statusValue = atoi (statusString);

// Read GPIB-error.
viRead (GPIB232, gpibERR, 10, &retCount);

// Read serial-error.
viRead (GPIB232, serialERR, 10, &retCount);

// Read count.
viRead (GPIB232, count, 10, &retCount);

// If error occurred
if (statusValue < 0)
{
    // Print GPIB-error and serial-error
    // values to determine what errors occurred.
    printf ("GPIB-error = %s\n", gpibERR);
    printf ("serial-error = %s\n", serialERR);
}
```

Example 2

```
// Turn on continuous status reporting,
// in numeric format.
char statCommand [] = "stat c n\r";
char statusString [100];
memset(statusString,0,100);
```

```
viWrite (GPIB232, statCommand, strlen(statCommand), &retCount);

// If we have 3 bytes in the serial port
// buffer, a typical response would be:
// 262<CR><LF>0<CR><LF>0<CR><LF>3<CR><LF>
// Read the GPIB-RS232/485/422 status; read 100
// bytes or until EOI is received.
viRead (GPIB232, statusString, 100, &retCount);
// Print the status information.
printf ("GPIB-RS232/485/422 status is: %s\n", statusString);
```

Example 3

```
// Turn on continuous status reporting,
// in symbolic format.
char statCommand [] = "stat c s\r";
char statusString [51];
viWrite (GPIB232, statCommand, strlen(statCommand), &retCount);
// Read the GPIB-RS232/485/422 status; read 50
// bytes or until EOI is received.
viRead (GPIB232, statusString, 50, &retCount);
statusString [retCount] = '\0';
// Print the status information.
printf ("GPIB-RS232/485/422 status is: %s\n", statusString);

Printed information is:
GPIB-RS232/485/422 status is :
CMPL
NGER
NSER
0
```

xon

Change Serial Port XON/XOFF Protocol

Type

Serial port function

Syntax

xon [booltx] [,boolrx]<CR>

Purpose

You can use xon at the beginning of your program to configure the GPIB-RS232/485/422 to communicate over the serial port using the same XON/XOFF protocol as your serial device.

Remarks

The argument booltx specifies whether to enable the XON/XOFF protocol when sending data out on the serial port. When booltx is enabled, the GPIB-RS232/485/422 monitors its serial receive buffer for XON/XOFF characters as it sends data over the serial port. If it receives the XOFF character (decimal 19 or <Ctrl-S>), it immediately stops sending data. When it receives the XON character (decimal 17 or <Ctrl-Q>), it begins sending data again. To send a data string that might contain a <Ctrl-S> or <Ctrl-Q>, you must disable booltx.

The argument boolrx specifies whether to enable the XON/XOFF protocol when receiving data over the serial port. When boolrx is enabled, as the GPIB-RS232/485/422 receives data over the serial port, it sends XOFF over the serial port when its serial port receive buffer is almost full. This tells the sender to stop sending data. When the GPIB-RS232/485/422 serial port receive buffer is ready to receive more bytes, the GPIB-RS232/485/422 sends XON over the serial port. This tells the sender to begin sending data again.



Note Enabling either booltx or boolrx will enable the XON/XOFF protocol for both sending and receiving the data to/from the serial port. This behavior is different from the GPIB-232-CT-A. Refer to Differences between the GPIB-232/485CT/CV-A and GPIB-RS232/485/422 for more information about this and other differences.

You should use XON/XOFF if your serial device does not recognize the hardware handshake protocol and you are transferring large amounts of data at high speeds. Without handshaking, there is danger of overflowing the serial port receive buffers of the GPIB-RS232/485/422 and serial device. When you enable XON/XOFF, the hardware

handshaking enabled by default is disabled. If you later decide to disable XON/XOFF, the hardware handshaking is re-enabled.

The power-on default of XON/XOFF for both sending and receiving is disabled.

If you call `xon` without an argument, the GPIB-RS232/485/422 returns the current handshake setting: `1` if the protocol is enabled or `0` if protocol is disabled.

The assignment `xon` makes remains in effect until you call `xon` again, call `onl`, or turn off the GPIB-RS232/485/422.

Example 1

```
// Enable GPIB-RS232/485/422 XON/XOFF protocol for TX and RX.
// Note that xon 0,1\r or xon 1,0\r would accomplish the same.
char xonCommand [] = "xon 1,1\r";
viWrite (GPIB232, xonCommand, strlen(xonCommand), &retCount);
```

Example 2

```
// Return current settings
char xonCommand [] = "xon\r";
char responseString [10];

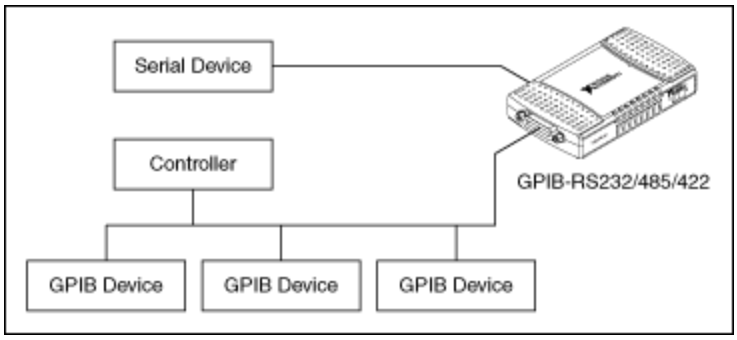
viWrite (GPIB232, xonCommand, strlen(xonCommand), &retCount);
viRead (GPIB232, responseString, 10, &retCount);

// responseString contains 1,1<CR><LF>
// transmit protocol enabled,
// receive protocol enabled.
```

D Mode Overview

In D Mode, the GPIB-RS232/485/422 acts as another GPIB device on the bus.

The following figure shows the D Mode system layout.



In D Mode, the GPIB-RS232/485/422 can act only as a GPIB Talker or Listener. Do not pass control to the GPIB-RS232/485/422 when using D Mode. If you do, the GPIB circuitry in the GPIB-RS232/485/422 accepts control and immediately asserts the ATN* signal, a system error condition you should avoid.

Because the GPIB-RS232/485/422 is not System Controller in D Mode, it expects to receive GPIB commands from the Controller-In-Charge (CIC). If the GPIB-RS232/485/422 receives the universal Device Clear (DCL) command or is addressed to listen and receives the Selected Device Clear (SDC) command, it clears its internal data buffers and resets to its power-up state. Neither Trigger (GET) nor Go To Local (GTL) have any effect on the GPIB-RS232/485/422.

Data Transfers in D Mode

When you are ready to send data to the GPIB-RS232/485/422 in D Mode, address the GPIB-RS232/485/422 to listen, if not already in listen-only mode. All GPIB data the GPIB-RS232/485/422 receives is buffered and passed along to the serial device. All serial data the GPIB-RS232/485/422 receives from the serial instrument is also buffered. When you are expecting to receive data from the serial device, address the GPIB-RS232/485/422 to talk, and all data received over the serial port is passed along to the GPIB port. The GPIB buffer size is 1 MB. The serial buffer size is 1 MB. The user cannot configure the sizes. This is a difference from the GPIB-232CV-A, which allowed the user to configure buffer sizes. For more information about this and other differences, refer to Differences between the GPIB-232/485CT/CV-A and GPIB-RS232/485/422.

Using the SRQ Feature

The GPIB bus performance may slow down if the Controller is constantly waiting for the GPIB-RS232/485/422 to send or receive data. With SRQ enabled, other devices can use the bus until the GPIB-RS232/485/422 requests service (asserts SRQ*). When the GPIB-RS232/485/422 asserts SRQ*, the Controller serial polls the GPIB-RS232/485/422 to determine what type of service it needs. After it is serial polled, the GPIB-RS232/485/422 unasserts SRQ*. You can enable the SRQ feature by running the NI GPIB-Serial Converter Wizard as described in Using the NI GPIB-Serial Converter Wizard and turning on the SRQ option as described in Configuration of D Mode.

Conditions that Cause SRQ to Be Asserted

If SRQ is enabled, the GPIB-RS232/485/422 asserts the SRQ* signal on two conditions:

- When the GPIB-RS232/485/422 receives serial data that it needs to pass on to the GPIB port.
- When the GPIB input buffer becomes empty.

When the GPIB-RS232/485/422 serial input buffer is empty and the GPIB-RS232/485/422 receives a serial character to be sent along to the GPIB, the SRQ* signal is asserted. This alerts the GPIB Controller that the GPIB-RS232/485/422 has data to send to a GPIB Listener. The Controller can then address the GPIB-RS232/485/422 to talk so that the data can be read out of the GPIB-RS232/485/422 serial buffer. When the serial buffer is empty, the Controller can unaddress the GPIB-RS232/485/422 to talk and address other devices to transfer data on the bus.

SRQ* is also asserted when the GPIB input buffer becomes empty. When the GPIB-RS232/485/422 has emptied the buffer, it requests service, indicating it is ready for more data. The Controller can then readdress the GPIB-RS232/485/422 to listen and continue sending data until all data is sent.

Maximizing Performance with SRQ

If large amounts of data (greater than the GPIB buffer size) are sent over the GPIB to the GPIB-RS232/485/422, and the serial device is a very slow data acceptor, the GPIB-RS232/485/422 internal GPIB data buffer might become full. If this occurs, the **FULL** LED on the front panel illuminates, and the GPIB transfer rate slows to approximately

the same rate as the serial device that is accepting the data. Because the maximum data transfer rate on the GPIB is only as fast as the slowest Listener, the GPIB system performance may become unacceptable if other GPIB devices are required to transfer data on the bus.

To get maximum GPIB performance when large amounts of data are sent to the GPIB-RS232/485/422, enable the SRQ feature and send only data amounts up to but not exceeding the GPIB input buffer size. The Controller can then unaddress the GPIB-RS232/485/422 to listen so that other devices can use the bus. When the GPIB-RS232/485/422 is ready to receive more data, it asserts the SRQ* signal.

GPIB-RS232/485/422 Serial Poll Responses

If the GPIB-RS232/485/422 is serial polled, its response byte depends on the internal data buffers' status, as follows:

- If the serial input buffer is empty, bit 1 of the status byte is set to 1.
- If the serial input buffer is not empty, bit 1 is set to 0.
- If the GPIB input buffer is empty, bit 0 of the status byte is set to 1.
- If the GPIB input buffer is not empty, bit 0 is set to 0.

In all cases, bit 6 is set to 1 if the GPIB-RS232/485/422 is requesting service; otherwise, it is set to 0.

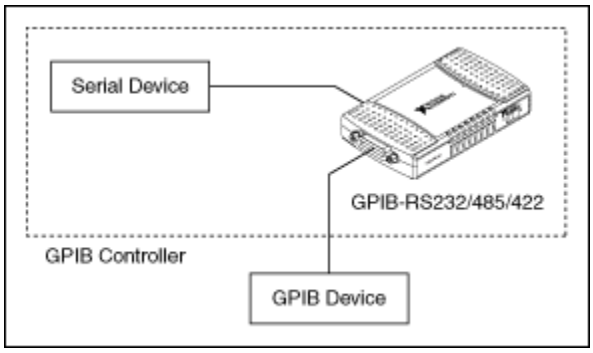
GPIB-RS232/485/422 Operation in Listen-Only Mode

In listen-only mode, no addressing is required to make the GPIB-RS232/485/422 a GPIB Listener, and it cannot be unaddressed to listen. All data transferred on the GPIB is accepted by the GPIB-RS232/485/422 and output to the serial port even if the data was intended for another device. You can use this configuration to monitor all GPIB data bus activity. Notice that when the GPIB-RS232/485/422 is configured as a listen-only device, it cannot be serial polled.

You can configure the GPIB-RS232/485/422 to enable listen-only mode by running the NI GPIB-Serial Converter Wizard as described in Using the NI GPIB-Serial Converter Wizard and turning on the Listen Only option as described in Configuration of D Mode.

C Mode Overview

In C Mode, the GPIB-RS232/485/422 and a serial host together act as the GPIB System Controller. The following figure shows the C Mode system layout.



When you turn on the power with the GPIB-RS232/485/422 in C Mode, the GPIB-RS232/485/422 asserts IFC* for 500 µs and addresses an instrument to listen. On a GPIB system, asserting IFC* for at least 100 µs initializes the GPIB and causes the System Controller to become Controller-In-Charge (CIC). After the GPIB-RS232/485/422 becomes CIC, it addresses the PAD/SAD configured through configuration mode as a listener. The GPIB-RS232/485/422 always asserts REN* when addressing an instrument.

Data Transfers in C Mode

All incoming serial data to the GPIB-RS232/485/422 is stored in an internal data buffer until the GPIB device accepts it. The internal data buffer that stores incoming serial data can hold a maximum of 1 MB. When the serial input buffer is full, the serial port stops accepting serial bytes. This results in blocking the host serial writes if Hardware or XonXoff flow control is used, or dropping data if no flow control is enabled. All GPIB data sent to the GPIB-RS232/485/422 is buffered in a GPIB input buffer that can hold a maximum of 1 MB. The GPIB-RS232/485/422 continuously attempts to send any data in the GPIB input buffer over the serial port, but if Hardware or XonXoff flow control is enabled and the host serial port is not accepting data, incoming GPIB data is buffered. If the GPIB input buffer becomes full, the GPIB-RS232/485/422 stops accepting GPIB data.

You can send data to your serial port as if you had a serial device connected to it. The

data is converted and sent to the GPIB instrument. When the instrument has data to send back, the GPIB-RS232/485/422 addresses the instrument to talk and passes all received data to the serial port.

Modes of Operation Within C Mode

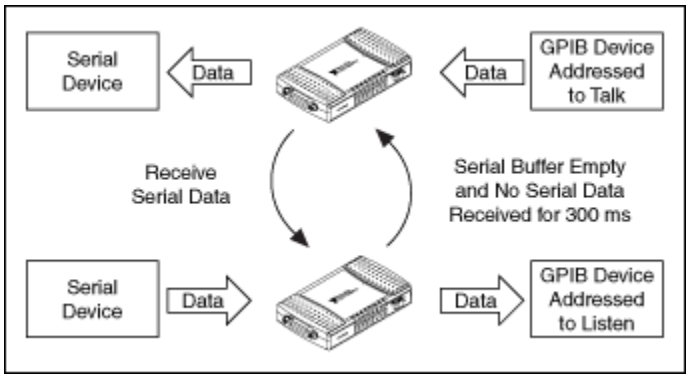
Four modes of operation are available within C Mode to define when the GPIB device is addressed to talk or listen. These modes are as follows:

- Timeout termination
- <CR> termination
- <LF> termination
- No-swap

In all modes, the instrument is addressed to listen after the GPIB-RS232/485/422 becomes CIC. All data received via the serial port is buffered and passed along to the device. The four modes of operation define the point at which the GPIB device is unaddressed to listen and addressed to talk so that data can be sent from the device to the GPIB-RS232/485/422.

Timeout Termination Mode

Timeout termination mode should be used for applications in which bidirectional communication is required, and neither a <CR> nor <LF> character is used to signal the end of a command string to the GPIB device.



In timeout termination mode, the GPIB-RS232/485/422 addresses the GPIB device to talk, and any characters read from the GPIB port are passed along immediately to the

serial port. When the GPIB-RS232/485/422 receives a serial character, it immediately readdresses the GPIB device to listen and passes the character to it. It remains in this mode until its serial input buffer becomes empty and no serial characters are received for 300 ms. After this period of inactivity, the GPIB-RS232/485/422 assumes that because no more serial data is being sent, the device might need to send data back to the GPIB-RS232/485/422. It then readdresses the device to talk and passes any characters read from the GPIB port along to the serial port.

<CR> or <LF> Termination Mode

If a command string to the GPIB instrument can be terminated with either a <CR> (carriage return) or <LF> (linefeed) character, you can use either termination with <CR> or <LF> to readdress the GPIB device to send data. In these modes, when the GPIB-RS232/485/422 receives the terminating character from the serial port, the serial input buffer is empty. When the device is addressed to talk, any serial character the GPIB-RS232/485/422 receives signals the GPIB-RS232/485/422 to readdress the GPIB device as a Listener.

Both <CR> and <LF> termination modes have the configurable option of asserting End Or Identity (EOI) on the GPIB bus on the terminating byte prior to readdressing the device as a Talker. To enable assertion of EOI, set EOI to ON through the configuration utility. To not assert EOI with <CR> or <LF>, set EOI to OFF through the configuration utility. EOI termination may be useful if your device requires EOI to be asserted to terminate commands and queries.

For details about the setup of this example, refer to General Programming Steps for Communicating with the GPIB-RS232/485/422 Over Serial.

```
//Actually performing the reads / writes here.
strcpy (stringinput,"FACTORY\n");
status = viWrite (instr, (ViBuf)stringinput, strlen(stringinput), &writeCount);
if (status < VI_SUCCESS)
{
printf ("Error writing to the device.\n");
}
strcpy (stringinput,"*IDN?\n");
status = viWrite (instr, (ViBuf)stringinput, strlen(stringinput), &writeCount);
if (status < VI_SUCCESS)
{
```

```
printf ("Error writing to the device.\n");
}

/*
Now we will attempt to read back a response from the device to
the identification query that was sent. We will use the viRead
function to acquire the data. We will try to read back 100 bytes.
This function will stop reading if it finds the termination character
before it reads 100 bytes.
After the data has been read the response is displayed.
*/
status = viRead (instr, buffer, 100, &retCount);
if (status < VI_SUCCESS)
{
printf ("Error reading a response from the device.\n");
}
else
{
printf ("\nData read: %*s\n", retCount, buffer);
}
```

No-Swap Mode

In no-swap mode, the GPIB-RS232/485/422 addresses the device as a listener. The GPIB-RS232/485/422 never readdresses a GPIB device after it has initially been addressed as a Listener. This mode is useful if your device is intended only to receive data. In the example in <CR> or <LF> Termination Mode, you could use the no-swap mode to talk to the plotter if you never needed to obtain status information from it. Because you cannot readdress in no-swap mode, the <CR> is not needed to obtain status as it was in the example in <CR> or <LF> Termination Mode.

For details about the setup of this example, refer to General Programming Steps for Communicating with the GPIB-RS232/485/422 Over Serial.

```
//Actually performing the reads / writes here.
strcpy (stringinput,"FACTORY\n");
status = viWrite (instr, (ViBuf)stringinput, strlen(stringinput), &writeCount);
if (status < VI_SUCCESS)
{
printf ("Error writing to the device.\n");
}
```

```
strcpy (stringinput,"LANGUAGE ENGLISH\n");
status = viWrite (instr, (ViBuf)stringinput, strlen(stringinput), &writeCount);
if (status < VI_SUCCESS)
{
printf ("Error writing to the device.\n");
}
```

Status and Error Message Information

The following topics describe the status and error information the GPIB-RS232/485/422 records as it executes each programming message. This information is returned in response to the `stat` command, or automatically after each command if continuous status reporting is enabled. Status is recorded in four parts: the 16-bit status word (`status`), the GPIB error indicator (`GPIB-error`), the serial error indicator (`serial-error`), and a count of the last I/O performed (`count`). For more information, refer to the S Mode or G Mode `stat` function description.

Status Bits

GPIB Error Codes

Serial Port Error Codes

Status Bits

The following paragraphs describe the conditions represented by the bits in `status`. The number preceding each description is the numeric value of that bit in the status word.

ERR	S or G Mode -32768
	The ERR bit is set in `status` following any call that results in an error. You can determine the particular error by examining the `GPIB-error` and `serial-error` values. The ERR bit is cleared following any call that does not result in an error. By examining this bit, you can check for an error condition after each call. An error made early in your application program might not become apparent until a later instruction. At that time, the error can be more difficult to locate.

TIMO	S Mode 16384
	The TIMO bit specifies whether a timeout has occurred. The TIMO bit is set in the status word following a call to `wait` if the TIMO bit of the wait `mask` parameter is also set and if the wait has exceeded the time limit value set by the `tmo` call. The TIMO bit is also set following a call to any of the I/O functions (for example, `rd`, `wrt`, and `cmd`), if a timeout occurs during a call. The TIMO bit is cleared in the status word in all other circumstances.
END	S Mode 8192
	The END bit specifies whether the END or EOS message has been received. The END bit is set in the status word following an `rd` function if the END or EOS message was detected during the read. While the GPIB-RS232/485/422 is performing a shadow handshake as a result of the `gts` function, any other function call can return a status word with the END bit set if the END or EOS message occurred before or during that call. The END bit is cleared in the status word at the start of any subsequent programming message.
SRQI	S Mode 4096
	The SRQI bit specifies whether a device is requesting service. This bit is set in the status word whenever the SRQ* line is asserted. The bit is cleared whenever the GPIB SRQ* line is unasserted.
CMPL	S or G Mode 256
	The CMPL bit specifies that the operation relating to this status information is complete. This bit is always set, and is useful in identifying the status word from other responses.
LOK	S Mode 128
	The LOK bit specifies whether the GPIB-RS232/485/422 is in a lockout state. The LOK bit is set whenever the GPIB-RS232/485/422 detects that either the GPIB-RS232/485/422 or another Controller has sent the Local Lockout (LLO) message. The LOK bit is cleared when the Remote Enable (REN*) GPIB line becomes unasserted. A call to `onl` also clears LOK.
REM	S Mode 64
	The REM bit specifies whether the GPIB-RS232/485/422 is in remote state. The REM bit is set whenever the Remote Enable (REN*) GPIB line is asserted and the GPIB-RS232/485/422 detects that either the GPIB-RS232/485/422 or another Controller has sent its listen address. The REM bit is cleared whenever REN* becomes unasserted, or when the GPIB-RS232/485/422 as a Listener detects the Go to Local (GTL) command, or when the `loc` function is called while the LOK bit is cleared in status. A call to `onl` also clears REM.
CIC	S Mode 32
	The CIC bit specifies whether the GPIB-RS232/485/422 is the Controller-In-Charge. The CIC bit is set whenever `sic` is called while the GPIB-RS232/485/422 is System Controller,

	or when another Controller passes control to the GPIB-RS232/485/422. The CIC bit is cleared whenever the GPIB-RS232/485/422 detects Interface Clear (IFC*) from some other device that is System Controller, or when the GPIB-RS232/485/422 passes control to another device. A call to `onl` also clears CIC.
ATN	S Mode 16
	The ATN bit specifies the GPIB Attention (ATN*) line state. The ATN bit is set whenever the GPIB ATN* line is asserted and cleared when the ATN* line is unasserted.
TACS	S Mode 8
	The TACS bit specifies whether the GPIB-RS232/485/422 has been addressed as a Talker. The TACS bit is set whenever the GPIB-RS232/485/422 detects that either the GPIB-RS232/485/422 itself or another Controller has sent its talk address (and secondary address, if enabled). The TACS bit is cleared whenever the GPIB-RS232/485/422 detects the Untalk (UNT) command, a talk address other than its own, its own listen address, or Interface Clear (IFC*). A call to `onl` also clears TACS.
LACS	S Mode 4
	The LACS bit specifies whether the GPIB-RS232/485/422 has been addressed as a Listener. The LACS bit is set whenever the GPIB-RS232/485/422 detects that either the GPIB-RS232/485/422 itself or another Controller has sent its listen address (and secondary address, if enabled). The LACS bit is also set whenever the GPIB-RS232/485/422 shadow handshakes as a result of the `gts` function. The LACS bit is cleared whenever the GPIB-RS232/485/422 detects that Unlisten (UNL), its own talk address, Interface Clear (IFC*), or `gts` is called without shadow handshake. A call to `onl` also clears LACS.
DTAS	S Mode 2
	The DTAS bit specifies whether the GPIB-RS232/485/422 has detected a device trigger command. The DTAS bit is set whenever the GPIB-RS232/485/422 as a Listener detects that another Controller has sent the Group Execute Trigger (GET) command. The DTAS bit is cleared in status at the start of any subsequent programming message.
DCAS	S Mode 1
	The DCAS bit specifies whether the GPIB-RS232/485/422 has detected a device clear command. The DCAS bit is set whenever the GPIB-RS232/485/422 detects that another Controller has sent the Device Clear (DCL) command, or whenever the GPIB-RS232/485/422 as a Listener detects that another Controller has sent the Selected Device Clear (SDC) command. The DCAS bit is cleared in status at the start of any subsequent programming message.

GPIB Error Codes

When the ERR bit is set in `status`, a GPIB error or a serial port error has occurred. The error code is indicated by `GPIB-error` or `serial-error`.

The following paragraphs describe the `GPIB-errors` in detail. The number preceding each description is the error code numeric value.

NGER	S or G Mode 0
	The GPIB-RS232/485/422 reports this error when the GPIB-RS232/485/422 detected no GPIB errors as a result of the last operation.
ECIC	S Mode 1
	The GPIB-RS232/485/422 records this value when you call a function that requires that the GPIB-RS232/485/422 be CIC and it is not CIC. In cases when the GPIB-RS232/485/422 should always be the Controller-In-Charge, call `sic` to send Interface Clear before attempting any of the calls, and avoid sending the command byte TCT (hex 09, Take Control). In multiple CIC situations, always be certain that the CIC bit appears in status before attempting the calls. If it does not appear, you can call `wait (256)` to delay further processing until control is passed to the GPIB-RS232/485/422.
ENOL	S Mode 2
	The ENOL error occurs most frequently when the GPIB-RS232/485/422 attempts to write data to the GPIB and no Listeners are addressed. To correct this error, be sure the proper listen address is in the `alist` argument string, use `cmd` to properly address the Listeners, or be sure some other Controller has addressed the Listeners before you call `wrt`. This error might occur less frequently in situations in which the GPIB-RS232/485/422 is not the CIC and the Controller asserts ATN* before the `wrt` call in progress has ended. In this case, either reduce the write byte count to that which the Controller expects, or resolve the situation on the Controller's end. This error also occurs during `cmd` if there is no device on the GPIB bus receiving the command bytes. In this case, check the GPIB cabling and verify that the attached GPIB devices are powered on.
EADR	S Mode 3
	The GPIB-RS232/485/422 records this error when it is not addressed to listen or to talk before `rd` and `wrt` calls when it is not CIC. Be sure the Controller addresses the GPIB-

	RS232/485/422 to talk or listen before attempting the `wrt` or `rd`. The GPIB-RS232/485/422 also records this error during the function `gts` when the shadow handshake feature is requested and the GPIB ATN* line is already unasserted. In this case, the shadow handshake is not possible and the error is recorded to notify you of that fact. `gts` should almost never be called except immediately after a `cmd` call. (`cmd` causes ATN* to be asserted.)
EARG	S or G Mode 4
	The GPIB-RS232/485/422 records this error when you pass an invalid argument to a function call. The following are some examples: • `tmo` called with a value not in the range .00001 to 3600 • `sic` called with a value not in the range .0001 to 3600 • `eos` called with meaningless termination method identifiers • `caddr` called with the value 31 • `ppc` called with illegal parallel poll configurations If your programming message contains more than one argument and you get this error, the GPIB-RS232/485/422 discards all arguments and does not perform the function. This error can also be caused by a transmission error that corrupts the argument portion of the programming message or the <CR> or <LF> that terminates which programming message. Use `stat` and check `serial-error` to determine if a transmission error has occurred.
ESAC	S Mode 5
	The GPIB-RS232/485/422 records this error when `sic` or `sre` is called when the GPIB-RS232/485/422 does not have System Controller capability. In this case, give the GPIB-RS232/485/422 Controller capability by calling `rsc`. (At power-on, the GPIB-RS232/485/422 assumes itself to be the System Controller.)
EABO	S Mode 6
	The GPIB-RS232/485/422 records this error when I/O has been canceled. The cause of this error is usually a timeout condition. If I/O is actually progressing but times out anyway, lengthen the timeout period with `tmo`. More frequently, however, either the Listener is not continuing to handshake, the Talker has stopped talking, or the byte count in the call that timed out was more than the other device was expecting. Be sure both parties to the transfer understand what byte count is expected. Or if possible, have the Talker use the END message to assist in early termination.
ECAP	S or G Mode 11

	This error results when a particular capability has been disabled in the GPIB-RS232/485/422 and a call is made that attempts to make use of that capability. A common cause of this error is that a programming message contains an S Mode function and the GPIB-RS232/485/422 is configured for G Mode, or a programming message contains a G Mode function and the GPIB-RS232/485/422 is configured for S Mode.
EBUS	S Mode 14
	This error indicates there was a problem sending command bytes out of the GPIB port. The most common causes of this error are either that the bytes could not be sent out within the timeout period, or that there was not a device on the GPIB bus to receive the command bytes. This error can occur during `clr`, `loc`, `pct`, `ppc`, `ppu`, `rd`, `rsp`, `trg`, or `wrt`.
ECMD	S or G Mode 17
	The GPIB-RS232/485/422 records this error when your programming message received does not contain a recognizable function name. This error can happen if the function name is misspelled or a transmission error occurred that resulted in the function name being corrupted. Check your function name spelling and check `serial-error` to see if a serial port error has been posted.

Serial Port Error Codes

The following paragraphs describe the serial port errors in detail. The number preceding each description is the error code numeric value.

In S Mode, when a serial port error occurs as the GPIB-RS232/485/422 receives a programming message, the error is recorded and the message is discarded. If a serial port error occurs during a data transfer such as `cmd`, `rd`, or `wrt`, that transfer is aborted. You can use the `spign` function to tell the GPIB-RS232/485/422 to ignore all serial port errors. When serial port errors are ignored, bytes that arrive with errors are not discarded, and no error is recorded in the serial port error code returned by `stat`.

NSER	0
	The GPIB-RS232/485/422 reports this value when there is no serial port error detected as a result of the last operation.
EPAR	1

	The GPIB-RS232/485/422 records this error when the parity of the received character is not what was expected; one or more bits of the received character were corrupted in a way that changed the character's parity.
EORN	2
	The GPIB-RS232/485/422 records this error when characters arrive at the serial port faster than the serial port can accept them. When this error occurs, one or more characters sent to the serial port have been lost. If this error occurs, check to see that the GPIB-RS232/485/422 and your serial device are using the same serial port settings.
EOFL	3
	The GPIB-RS232/485/422 records this error when the GPIB-RS232/485/422 internal serial port buffer overflows. This error should occur only if XON/XOFF is disabled and no hardware handshake is in effect.
EFRM	4
	The GPIB-RS232/485/422 records this error when a character is received whose stop bits are not in the expected place. This error can happen when the number-of-bits-per-character setting of the GPIB-RS232/485/422 does not match your serial device. It can also happen if the baud rates of the GPIB-RS232/485/422 and your serial device do not match, or if one side of the serial link does not use parity and the other side does. The GPIB-RS232/485/422 also records this error when it receives a serial break from the attached serial device. The reception of a serial break is detected when the Receive Data (RXD) line is unasserted for longer than a full word transmission time (that is, the total time of start bit + data bits + parity + stop bits).

Interfacing to an RS232 Device

The following topics describe the RS232 serial port on the GPIB-RS232 and explain how to interface a DCE or DTE serial device to the RS232 serial port.

The GPIB-RS232 transfers serial data using the electrical signals, mechanical connections, data format, and control protocols defined in the ANSI/EIA-232-D (RS232) standard. The RS232 port on the GPIB-RS232 provides an asynchronous serial communication link to a serial peripheral device.

RS232 Standard

Description of the RS232 Port

DTE vs. DCE

Interfacing Serial Devices to the RS232 Serial Port

Interfacing the GPIB-RS232 to a DCE with Handshaking

Interfacing the GPIB-RS232 to a DCE without Handshaking

Minimum Configuration Cable

Custom Cables

Interfacing the GPIB-RS232 to a DTE with Handshaking

Interfacing the GPIB-RS232 to a DTE without Handshaking

Minimum Configuration Cable

Custom Cables

RS232 Standard

RS232, as specified in the ANSI/EIA-232-D Standard, ***Interface Between Data Terminal Equipment and Data Circuit-Terminating Equipment Employing Serial Binary Data Interchange***, standardizes serial communication between computers and between computer terminals and modems. Most applications use the RS232 standard for interfacing peripherals to personal computers. RS232 uses transmission lines in which the each signal state is represented by referencing the voltage level of a single line to ground. RS232 was designed for serial communication up to distances of 50 ft and with data rates up to 20 kbytes/s. However, because of improvements in line drivers and cabling, you can often increase the actual performance of the bus past the limitations on speed and distance recommended in the specification. For more information about the RS232 standard, contact the following organization:

Global Engineering Documents
15 Inverness Way East

Englewood, CO 80112
(800) 854-7179
global.ihs.com

Description of the RS232 Port

The RS232 serial port on the GPIB-RS232 uses a male 9-pin D-Subminiature connector with a DTE interface configuration. The following table shows the signal lines supported on the GPIB-RS232.

RS232 Serial Port Signal Configuration

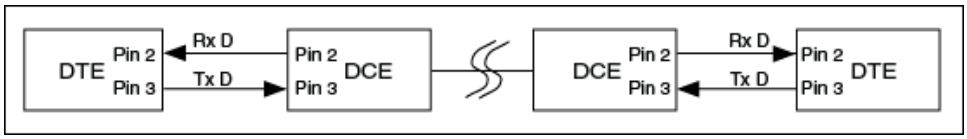
Pin Number	Signal Description	RS232 Code	Function
2	RXD (Receive Data)	BB	This signal carries serial data from the serial device to the GPIB-RS232.
3	TXD (Transmit Data)	BA	This signal carries serial data from the GPIB-RS232 to the serial device.
4	DTR (Data Terminal Ready)	CD	This signal is asserted by the GPIB-RS232 to signal that it has been powered on and is ready to operate.
5	GND (Ground)	AB	This signal establishes a reference point for all interface voltages.
7	RTS (Request to Send)	CA	This signal is driven by the GPIB-RS232. When asserted, it indicates the GPIB-RS232 is ready to accept serial data. When unasserted, it indicates that the GPIB-RS232 is no longer ready to accept serial data because the buffer is full.
8	CTS (Clear to Send)	CB	This signal is sensed by the GPIB-RS232. When asserted, it indicates the serial device is ready to accept serial data. When unasserted, it indicates that data transmission should be disabled.

DTE vs. DCE

Data Terminal Equipment (DTE) and Data Communications Equipment (DCE) were the

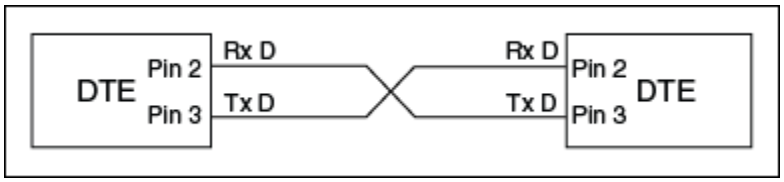
terms used in the RS232 specification for the types of equipment on either end of a serial connection. (A DCE is now called Data Circuit-Terminating Equipment in Revision D of the RS232 specification.) In general, DTE and DCE refer to computer equipment and modems respectively. Because the RS232 specification mainly involved connecting a DTE directly to a DCE and vice versa, the pinouts were defined so that cabling was simple. That is, a cable connected a computer to a modem by wiring pin 1 to pin 1, pin 2 to pin 2, and so on. This method is commonly known as straight-through cabling.

The following figure shows straight-through cabling in a DTE-to-DCE interface.



Straight-through cabling is still the standard method to connect a modem to your PC. However, because many applications use serial communication to connect two or more DTEs without modems, the cabling becomes more complicated. If two DTEs were wired together using a straight-through cable, one transmitter would be connected to the other transmitter, and one receiver would be connected to the other receiver. In this setup, no transmissions could occur. Thus, these applications must use a cabling scheme that connects the transmitter on one device to the receiver on the other device, and vice versa. This method is known as null-modem cabling, because it replaces the two modems that traditional RS232 applications would require between the two DTEs. You should use a null-modem cable to communicate from one AT serial port to another.

The following figure shows null-modem cabling in a DTE-to-DCE interface.



Interfacing Serial Devices to the RS232 Serial Port

To interface other serial devices to the RS232 serial port on the GPIB-RS232, first refer to the manual that came with your serial device to determine if the device is

configured as a DTE or DCE. Also, determine from the manual how the control lines are used and whether they must be driven for the serial port to operate.

Note The GPIB-RS232 serial port is configured to be a DTE.

Interfacing the GPIB-RS232 to a DCE with Handshaking

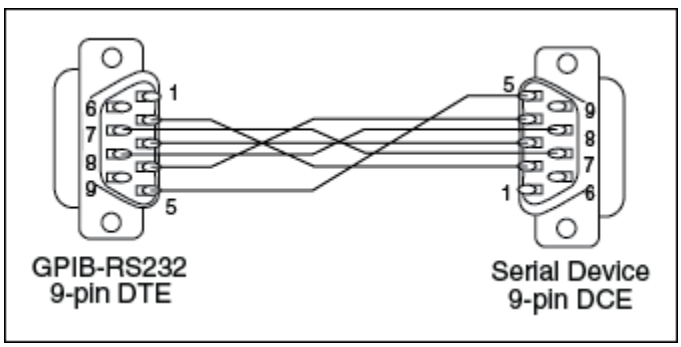
A correctly configured DTE-to-DCE interface is wired straight across: GPIB-RS232 TXD to DCE TXD, GPIB-RS232 RXD to DCE RXD, and so on, as shown in the following table.

Cable Wiring Scheme for GPIB-RS232 DTE to Serial Device DCE

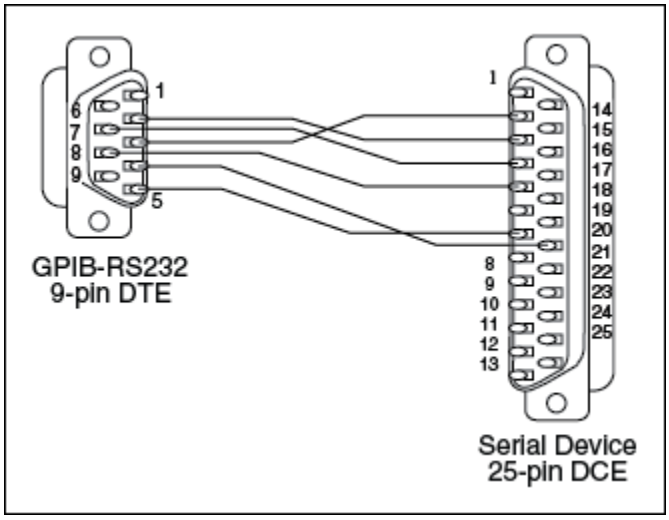
GPIB-RS232 Signal to DCE Serial Device Signal	Standard DTE to DCE 9-pin 9-pin	Standard DTE to DCE 9-pin 25-pin
RXD to RXD[1]	**2 to 2**	**2 to 3**
TXD to TXD[1]	**3 to 3**	**3 to 2**
DTR to DTR	4 to 4	4 to 20
GND to GND[1]	**5 to 5**	**5 to 7**
RTS to RTS	7 to 7	7 to 4
CTS to CTS	8 to 8	8 to 5

[1]The connections must be implemented.

The following figure shows a properly configured 9-pin DTE to 9-pin DCE cable including the hardware handshake lines RTS, CTS, and DTR. With this configuration, the GPIB-RS232 can function properly (handshake) on buffer full conditions.



The following figure shows an equivalent 9-pin DTE to 25-pin DCE cable configuration.



Interfacing the GPIB-RS232 to a DCE without Handshaking

If your serial device does not require or use the hardware handshaking protocol used by the GPIB-RS232, you have two options:

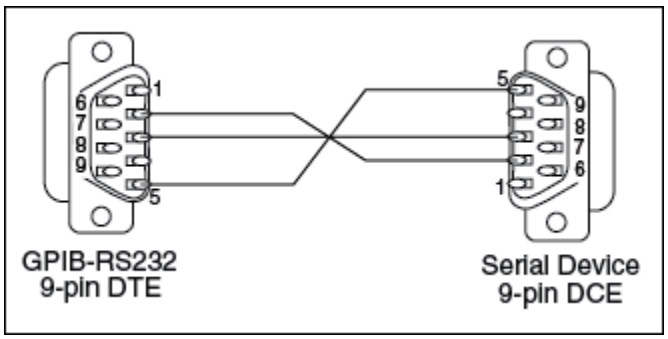
- Use a minimum configuration cable that does not support hardware handshaking and use XON/XOFF software handshaking (if necessary). To create a minimum configuration cable, connect the signals shown in bold in the following table.

GPIB-RS232 Signal to DCE Serial Device Signal	**Standard DTE to DCE 9-pin 9-pin**	**Standard DTE to DCE 9-pin 25-pin**
RXD to RXD[1]	**2 to 2**	**2 to 3**
TXD to TXD[1]	**3 to 3**	**3 to 2**
DTR to DTR	4 to 4	4 to 20
GND to GND[1]	**5 to 5**	**5 to 7**
RTS to RTS	7 to 7	7 to 4
CTS to CTS	8 to 8	8 to 5
[1]The connections must be implemented.		

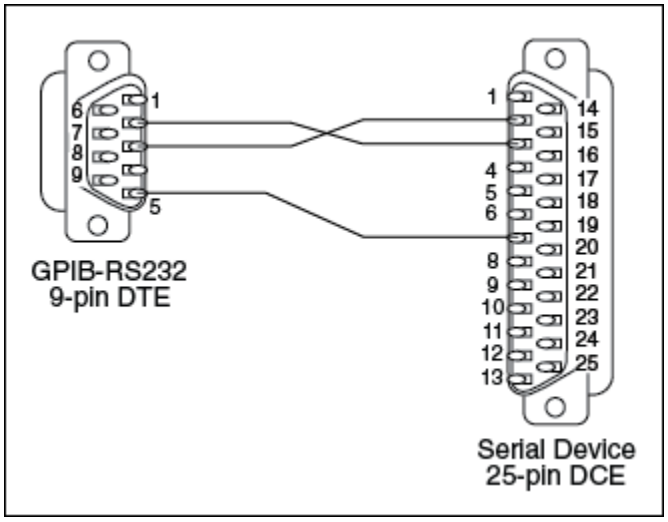
- Wire a custom cable that properly interfaces the GPIB-RS232 hardware handshaking protocol to the handshaking protocol of your serial device.

Minimum Configuration Cable

The minimum configuration cable assumes that the DCE does not require external hardware handshaking. The following figure shows the minimum configuration for a DTE 9-pin to DCE 9-pin cable.



The following figure shows an equivalent 9-pin to 25-pin cable.



Custom Cables

If your application requires a custom cable, you can construct one if you have a thorough knowledge of the handshaking protocols involved. Review the RS232 characteristics of your serial device and build the cable to properly connect the handshake lines of the two devices. Because the DTE-to-DCE connection is a straight-

across connection, it often involves only connecting RTS to RTS, CTS to CTS, DTR to DTR, and so on. If the documentation for your serial device does not provide a thorough explanation of its handshaking protocol, the ANSI/EIA-232-D standard is a good reference, provided your device conforms to the RS232 protocol.

Caution Although handshaking might not be required, it is best to use some form of handshaking to prevent loss of data.

Interfacing the GPIB-RS232 to a DTE with Handshaking

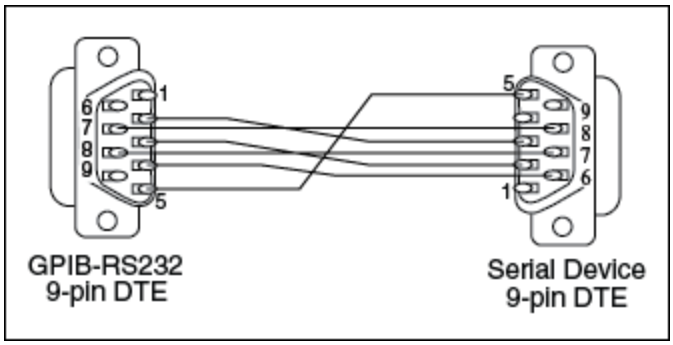
For serial devices set up as DTEs, you must wire a DTE-to-DTE interface cable, commonly known as a null modem cable. The cable allows the GPIB-RS232 to act as if it is communicating with a DCE, but it swaps the appropriate pins to achieve a DTE configuration. This wiring configuration is shown in the following table.

Cable Wiring Scheme for GPIB-RS232 DTE to Serial Device DTE

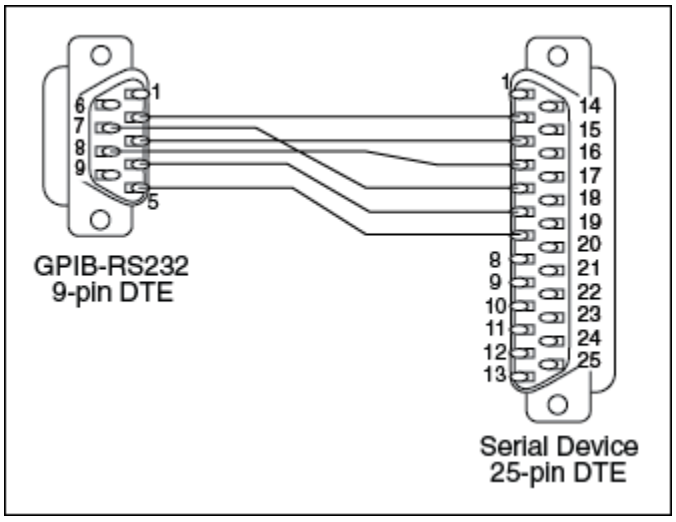
GPIB-RS232 Signal to DTE Serial Device Signal	**Standard DTE to DTE 9-pin 9-pin**	**Standard DTE to DTE 9-pin 25-pin**
RXD to TXD[1]	**2 to 3**	**2 to 2**
TXD to RXD[1]	**3 to 2**	**3 to 3**
DTR to DSR	4 to 6	4 to 6
GND to GND[1]	**5 to 5**	**5 to 7**
RTS to CTS	7 to 8	7 to 5
CTS to RTS	8 to 7	8 to 4

[1]The connections must be implemented.

The following figure shows a typical 9-pin to 9-pin null modem cable with the RTS, CTS, and DTR handshake lines implemented. This cable configuration allows you to connect to the 9-pin serial port of a personal computer.



The following figure shows an equivalent 9-pin DTE to 25-pin DTE cable configuration. The cable in this figure shows how to connect to the 25-pin serial port of a personal computer.



Interfacing the GPIB-RS232 to a DTE without Handshaking

If your serial device does not require or use the same hardware handshaking protocol used by the GPIB-RS232, you have two options:

- Use a minimum configuration null modem cable that does not support the hardware handshake lines and use XON/XOFF software handshaking (if necessary).

  To create a minimum configuration cable, connect the signals shown in bold in the following table.

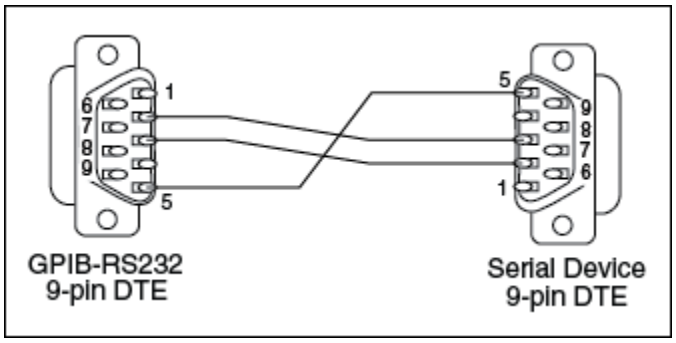
GPIB-RS232 Signal to DTE Serial Device Signal	Standard DTE to DTE	Standard DTE to DTE

	9-pin 9-pin	9-pin 25-pin
RXD to TXD[1]	**2 to 3**	**2 to 2**
TXD to RXD[1]	**3 to 2**	**3 to 3**
DTR to DSR	4 to 6	4 to 6
GND to GND[1]	**5 to 5**	**5 to 7**
RTS to CTS	7 to 8	7 to 5
CTS to RTS	8 to 7	8 to 4
[1]The connections must be implemented.		

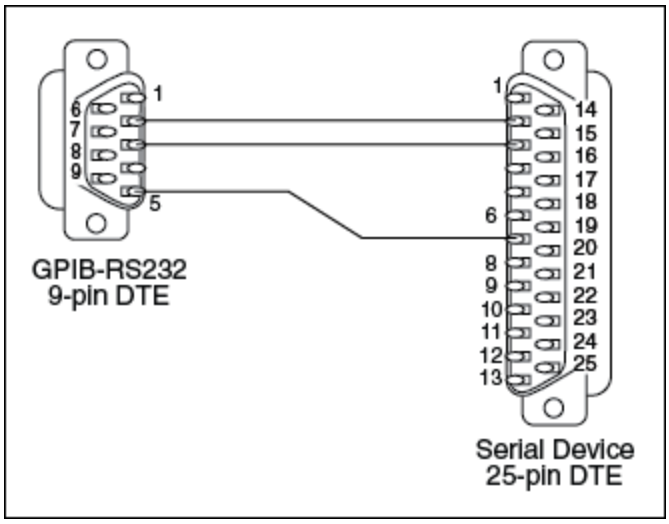
- Wire a custom cable that properly interfaces the GPIB-RS232 hardware handshaking protocol to the handshaking protocol of your serial device.

Minimum Configuration Cable

The minimum cable configuration assumes that the serial device does not require external hardware handshaking. The following figure shows the minimum configuration for a 9-pin to 9-pin null modem cable.



The following figure shows an equivalent 9-pin to 25-pin cable.



Custom Cables

If your application requires a custom cable, you can construct one if you have a thorough knowledge of the handshaking protocols involved. Review the RS232 characteristics of your serial device and build the cable to properly connect the handshake lines of the two devices. If the documentation for your serial device does not provide a thorough explanation of its handshaking protocol, the ANSI/EIA-232-D standard is a good reference, provided your device conforms to the RS232 protocol.

Caution Although handshaking might not be required, it is best to use some form of handshaking to prevent loss of data.

Interfacing to an RS485 Device

This topic describes the RS485 serial port on the GPIB-RS485/RS422 and explains how to interface an RS485 device to the RS485 serial port.

The GPIB-RS485/RS422 transfers serial data using the electrical signals defined in the EIA-485 standard. The RS485 port on the GPIB-RS485/RS422 provides an asynchronous serial communication link to a serial peripheral device. For more information on the EIA-485 standard, contact the following organization:

Global Engineering Documents
15 Inverness Way East

Englewood, CO 80112
(800) 854-7179
global.ihs.com

RS422

RS422

RS422, as specified in the EIA/RS422-A Standard, ***Electrical Characteristics of Balanced Voltage Digital Interface Circuits***, defines a serial interface much like RS232. However, RS422 uses balanced (or differential) transmission lines. Balanced transmission lines use two transmission lines for each signal. The each signal state is represented not by a voltage level on one line referenced to ground as in RS232, but rather by the relative voltage of the two lines to each other. For example, the TX signal is carried on two wires, wire A and wire B. A logical 1 is represented by the voltage on line A being greater than the voltage on line B. A logical 0 is represented by the voltage on line A being less than the voltage on line B. Differential voltage transmission creates a signal that is much more immune to noise as well as voltage loss due to transmission line effects. Thus, you can use RS422 for much longer distances (up to 4,000 ft) and much greater transmission speeds (up to 10 Mbytes/s) than RS232.

GPIB Basics

This topic contains information about how the GPIB operates, including information about GPIB messages, the roles of Talkers, Listeners, and Controllers, GPIB signals and lines, and connecting and configuring devices on the bus.

The ANSI/IEEE Standard 488.1-1987, also known as GPIB (General Purpose Interface Bus), describes a standard interface for communication between instruments and controllers from various vendors. It contains information about electrical, mechanical, and functional specifications. The GPIB is a digital, 8-bit parallel communications interface with data transfer rates of 1 Mbyte/s and above. The bus supports one System Controller, usually a computer, and up to 14 additional instruments. The ANSI/IEEE Standard 488.2-1992 extends IEEE 488.1 by defining a bus communication protocol, a common set of data codes and formats, and a generic set of common device commands.

Types of GPIB Messages

Types of GPIB Messages

The GPIB carries device-dependent messages and interface messages.

Device-dependent messages, often called data or data messages, contain device-specific information such as programming instructions, measurement results, machine status, and data files.

Interface messages manage the bus. They are usually called commands or command messages. Interface messages perform such tasks as initializing the bus, addressing and unaddressing devices, and setting device modes for remote or local programming. Multiline Interface Messages lists the GPIB command messages.

Parallel Polling

The following topics explain the use and operation of parallel polls. By using parallel polls, a GPIB Controller can obtain information from several devices on the GPIB in one operation. The Controller polls configured devices and reads back a single response byte that contains one bit of information from each device. From this information, the Controller can determine which devices need service.

Operation

Configuration

Issuing Remote Configurations in S Mode

Issuing Local Configurations in S Mode

The Parallel Poll

S Mode Parallel Polling Examples

Example 1

Example 2

Operation

The GPIB Controller can conduct a parallel poll at any time. To execute the poll, the Controller sends the IDY* (identify) message on the bus. A device configured for parallel polls responds at this time by driving an assigned data line either TRUE or FALSE, depending on the value of its individual status (ist) bit. When a parallel poll is conducted, the device determines the value of its ist bit and drives the line to the appropriate value. Whether this line is TRUE or FALSE depends on how the device is configured for the poll.

The circumstances under which a device sets its ist bit are specific to the device. For example, a device might always set its ist bit to 1 when it is busy and 0 when it is free, or vice versa. Consult your device documentation for this information.

There are two steps to conducting a parallel poll:

- Configuration—sets up the devices to participate in the poll.
- The parallel poll—reads the data lines and reports the result.

The following topics describe these two steps:

Configuration

Issuing Remote Configurations in S Mode

Issuing Local Configurations in S Mode

Configuration

There are two methods for configuring a device to respond to parallel polls. A device may use only one method.

- The device configures itself. This is referred to in the IEEE 488 specification as Parallel Poll (PP) Interface Function Subset PP2, or local configuration.
- The device allows an external Controller to configure it. This is referred to in the

IEEE 488 specification as PP Interface Function Subset PP1, or remote configuration.

In S Mode, you can choose which configuration method to use by setting or clearing the PP2 option of the `conf` function. When the PP2 option is cleared (the default), the GPIB-RS232/485/422 uses PP subset PP1. This causes the GPIB-RS232/485/422 to accept only those configurations that come over the GPIB from an external Controller. When the PP2 option is set, the GPIB-RS232/485/422 uses PP subset PP2. This causes the GPIB-RS232/485/422 to allow local configurations and ignore configurations that come over the GPIB from an external Controller.

In G mode, PP subset PP1 is the only allowable parallel poll configuration method. Thus, the GPIB Controller in your system must configure the GPIB-RS232/485/422 for it to respond to parallel polls. Parallel poll configurations are accomplished by using Parallel Poll Enable (PPE) messages and Parallel Poll Disable (PPD) messages. There are 16 possible PPE messages—hex 60 through hex 6F. There are also 16 possible PPD messages—hex 70 through hex 7F. The following figure shows the meaning of the bits in the PPE and PPD messages.

0	1	1	U	S	DIO lines 1 though 8		
			X	X	X	X	X

Parallel Poll Message (PPE or PPD) Bits

The following table contains a list of the parallel poll message bits and a description of each bit. Because the `U` bit is set in all PPD messages, they all have the same effect. All PPD messages disable the device from responding to parallel polls.

Parallel Poll Message Bit Descriptions

Bit	Description
U	If 0 (hex 6X), parallel poll is enabled. If 1 (hex 7X), parallel poll is disabled.
S	If the `ist` (individual status) bit of the device matches the `S` bit, the device sets the appropriate data line. Hex 60 through hex 67, set `S` to 0; hex 68 through hex 6F, set `S` to 1.
DIO	The value n in bits 0 through 2 corresponds to one of the DIO lines 1 through 8, where n corresponds to DIO line n`+1`. Thus, a value of 2 (binary 010) corresponds to DIO line 3.

Issuing Remote Configurations in S Mode

In S Mode, the `ppc` and `ppu` functions configure devices remotely for parallel polls. Remote configuration is indicated by passing the addresses of the GPIB devices to be configured in the `addr` arguments. For example, to configure a device at address 5 to respond on DIO line 3 when its `ist` bit is 1, send the following programming message to the GPIB-RS232/485/422:

```
ppc 5,3,1<CR>
```

The following figure shows the PPE message formed by the `ppc` programming message given above.

0	1	1	U	S	DIO lines 1 though 8		
			0	1	0	1	0

Sample PPE Message Bits

The value of this byte is hex 6A, where:

- `U` = 0 Enable parallel poll responses
- `S` = 1 When `ist` = 1, the device asserts DIO line 3 (which corresponds to 010 in bits 0 through 2) in response to a parallel poll.

The `ppc` function enables GPIB devices to respond to parallel polls, and the `ppu` function disables parallel poll responses. For every `addr, ppr, s` configuration, the `ppc` function sends the talk address of the GPIB-RS232/485/422, unlisten, the listen address specified by `addr`, Parallel Poll Configure, then the Parallel Poll Enable message (PPE). The PPE message is formed by using `ppr` as the DIO line and `s` as the `S` bit. For every `addr` argument given, the `ppu` function sends the talk address of the GPIB-RS232/485/422, unlisten, the listen address specified by `addr`, Parallel Poll Configure, then the Parallel Poll Disable (PPD) message. The PPD message disables only one GPIB device at a time from responding to parallel polls.

If the `ppu` function is called without an argument, it disables all GPIB devices from responding to parallel polls. It does this by sending out the Parallel Poll Unconfigure (PPU) message. This message has the same effect as using the `ppu` function to send

PPD messages to every device on the GPIB.

For the ppc or ppu function to have an effect, the device it is configuring must be using PP subset PP1. If the device is not using PP subset PP1, it does not allow an external Controller to configure it, and will probably ignore the configuration.

Issuing Local Configurations in S Mode

In S Mode, the ppc and ppu functions also configure the GPIB-RS232/485/422 locally. Local configuration is indicated by sending one addr argument with the value 255. For example, to configure the GPIB-RS232/485/422 to respond on DIO line 5 when its ist bit is 0, send the following programming message to the GPIB-RS232/485/422:

```
ppc 255,5,0<CR>
```

When configuring the GPIB-RS232/485/422 itself, the ppc or ppu function does not send messages out on the GPIB. Instead, the GPIB-RS232/485/422 internally enables or disables parallel poll responses as indicated. For the ppc or ppu function to have an effect, the GPIB-RS232/485/422 must be using PP subset PP2. If the GPIB-RS232/485/422 is not configured to use PP subset PP2, it cannot allow itself to be locally configured, and returns an ECAP error. You can configure the GPIB-RS232/485/422 to use PP subset PP2 with the conf function.

The Parallel Poll

In S Mode, after parallel poll configuration is complete, the GPIB-RS232/485/422 conducts a parallel poll by calling rpp. In the previous example, where the device was remotely configured using a PPE message of hex 6A, if the ist bit of the device is set, rpp returns a value of hex 04. Here, the third least significant bit is set, corresponding to DIO line 3. (If any other devices responded positively on other lines, those corresponding bits would be set also.)



Note You can configure more than one device to respond on the same data line, in which case the bits in the response byte are set by ORing all the responses on that line. In G Mode, the GPIB-RS232/485/422 sets its ist bit whenever it asserts SRQ*, and clears it whenever it unasserts SRQ*. Refer to the srqen function description for the conditions under which the GPIB-RS232/485/422 asserts SRQ*. If the Controller remotely configures the GPIB-RS232/485/

	422 using a PPE message of hex 6D (binary 0110 1101) and parallel polls it while it is asserting SRQ*, it responds by asserting DIO line 6. If the Controller remotely configures the GPIB-RS232/485/422 using a PPE message hex 65 (binary 0110 0101) and parallel polls it while it is asserting SRQ*, it responds by not asserting DIO line 6.

S Mode Parallel Polling Examples

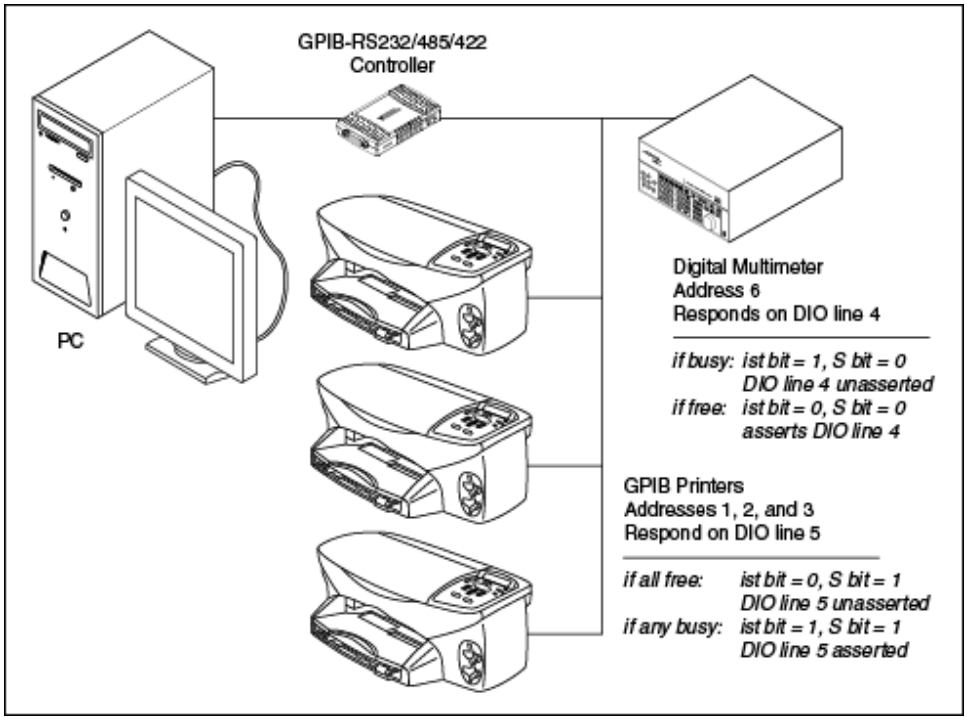
An example system has three line printers, one digital multimeter, and one PC with a serial port. The PC uses a GPIB-RS232/485/422 to communicate on the GPIB. The GPIB-RS232/485/422 is the designated Controller, and all other devices recognize it as the Controller. All GPIB devices set their `ist` bit to 1 when they are busy and 0 when they are free. Furthermore, all GPIB devices use PP subset PP1 (remote configuration by the Controller).

Example 1

Example 2

Example 1

The following figure shows an example parallel polling situation. The following paragraphs describe the configuration and responses.



The PC configures the multimeter (at address 6) to respond positively on DIO line 4 when free by sending the following programming message to the GPIB-RS232/485/422:

```
ppc 6,4,0<CR>
```

This programming message causes a PPE message of hex 63 to be sent to the multimeter. When a parallel poll is conducted by calling `rpp`, one of two things happens. If the multimeter is free, it asserts DIO line 4 (`rpp` returns hex 8); if it is busy, it unasserts DIO line 4 (`rpp` returns hex 0). When the multimeter is free, its `ist` bit is 0, and because this equals the value of the `S` bit in the PPE message, the device asserts DIO line 4.

The PC configures all line printers (at addresses 1, 2, and 3) to respond positively on DIO line 5 when busy by sending the following programming message to the GPIB-RS232/485/422:

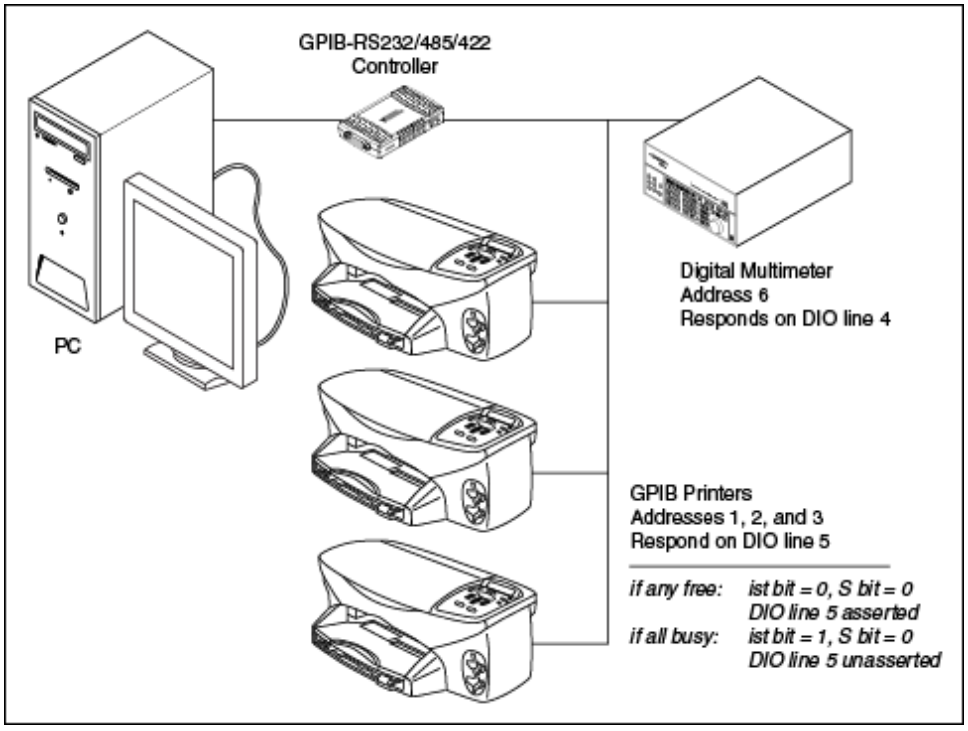
```
ppc 1,5,1 2,5,1 3,5,1<CR>
```

This programming message sends a PPE message of hex 6C to each line printer. When a parallel poll is conducted by calling `rpp`, the PC can find out immediately if any printer is busy or if all printers are free. If all line printers are free, `rpp` returns hex 0 (or hex 8 if the multimeter is free). When a line printer is free, its `ist` bit is 0, and because

this does not equal the value of the S bit in the PPE message, the device unasserts DIO line 5. If one or more line printers is busy, rpp returns hex 10 (or hex 18 if the multimeter is free). When a line printer is busy, its ist bit is 1, and because this equals the value of the S bit in the PPE message, the device asserts DIO line 5.

Example 2

Instead of checking to see if any printer is busy, you may want to know if any printer is free. The following figure shows a parallel polling setup to accomplish this. The following paragraphs describe the configuration and responses.



To find out if any printer is free, you must change the S bit/ist bit correspondence by reconfiguring the line printers with the following programming message:

```
ppc 1,5,0 2,5,0 3,5,0<CR>
```

This programming message sends a PPE message of hex 68 to each line printer. When a parallel poll is conducted by calling rpp, the PC can find out if any line printer is free or if all line printers are busy.

Specifications

This topic includes the GPIB-RS232/485/422 physical, electrical, and environmental characteristics.

Physical

Dimensions	16.01 x 9.35 x 3.15 cm (6.3 x 3.68 x 1.24 in.)
Case material	PC-ABS plastic
Weight	
RS232	192 g (6.75 oz)
RS485/RS422	196 g (6.875 oz)
GPIB cable	Type X2 shielded

Connectors

GPIB	IEEE 488 24-pin
RS232	DB-9 male
RS485/RS422	DB-9 male
DC power	Coaxial plug (single output models)  5.5 mm outer diameter 2.1 mm inner diameter, female 11 mm length, center "+"

Signaling

GPIB	3-wire
RS232/RS485	Baud rates up to 115.2 kb/s

Power Requirements

Input voltage range	9 to 28 VDC
Current consumption at 12 VDC	300 mA typical, 800 mA maximum
Fuse rating and type	Fast acting 2.2 A 125 V, surface mount

12 VDC Power Supply

Input voltage range	100 to 240 VAC, 47 to 63 Hz

Environment

Maximum altitude	2,000 m (at 25° C ambient temperature)
Pollution Degree	2
Indoor use only	

Operating Environment

Note For the GPIB-RS232/485/422 to operate correctly over the entire specified ambient temperature range, stacking the product is not recommended.

Ambient temperature range	0 to 55° C (Tested in accordance with IEC-60068-2-1 and IEC-60068-2-2.)
Relative humidity range	10 to 90%, noncondensing (Tested in accordance with IEC-60068-2-56.)

Storage Environment

Ambient temperature range	–20 to 70° C (Tested in accordance with IEC-60068-2-1 and IEC-60068-2-2.)
Relative humidity range	5 to 95%, noncondensing (Tested in accordance with IEC-60068-2-56.)

Shock and Vibration

Operational shock	30 g peak, half-sine, 11 ms pulse (Tested in accordance with

	IEC-60068-2-27. Test profile developed in accordance with MIL-PRF-28800F.)
Random vibration	
Operating	5 to 500 Hz, 0.3 g_{rms}
Nonoperating	5 to 500 Hz, 2.4 g_{rms} (Tested in accordance with IEC-60068-2-64. Nonoperating test profile exceeds the requirements of MIL-PRF-28800F, Class 3.)

Safety

This product is designed to meet the requirements of the following standards of safety for information technology equipment:

- IEC 60950-1, EN 60950-1
- UL 60950-1, CAN/CSA C22.2 No. 60950-1

Note For UL and other safety certifications, refer to the product label or visit `ni.com/certification`, search by model number or product line, and click the appropriate link in the **Certification** column.

Note The protection provided by this equipment may be impaired if it is used in a manner not described in this document.

Electromagnetic Compatibility

This product is designed to meet the requirements of the following standards of EMC for electrical equipment for measurement, control, and laboratory use:

- EN 61326 EMC requirements; Minimum Immunity
- EN 55011 Emissions; Group 1, Class A
- CE, C-Tick, ICES, and FCC Part 15 Emissions; Class A

Note For EMC compliance, operate this device according to product documentation.

CE Compliance

This product meets the essential requirements of applicable European Directives, as amended for CE marking, as follows:

- 73/23/EEC; Low-Voltage Directive (safety)
- 89/336/EEC; Electromagnetic Compatibility Directive (EMC)

Note Refer to the Declaration of Conformity (DoC) for this product for any additional regulatory compliance information. To obtain the DoC for this product, visit `ni.com/certification`, search by model number or product line, and click the appropriate link in the **Certification** column.

Waste Electrical and Electronic Equipment (WEEE)

EU Customers At the end of their life cycle, all products must be sent to a WEEE recycling center. For more information about WEEE recycling centers and National Instruments WEEE initiatives, visit `ni.com/environment/weee.htm`.

Multiline Interface Messages

This topic lists the multiline interface messages and describes the mnemonics and messages that correspond to the interface functions. The multiline interface messages are commands the IEEE 488 standard defines. The messages are sent and received with ATN asserted. The interface functions include initializing the bus, addressing and unaddressing devices, and setting device modes for local or remote programming. For more information about these messages, refer to the ANSI/IEEE Standard 488.1-1987, IEEE Standard Digital Interface for Programmable Instrumentation.

Hex	Oct	Dec	ASCII	Msg
00	000	0	NUL	
01	001	1	SOH	GTL
02	002	2	STX	
03	003	3	ETX	

04	004	4	EOT	SDC
05	005	5	ENQ	PPC
06	006	6	ACK	
07	007	7	BEL	
08	010	8	BS	GET
09	011	9	HT	TCT
0A	012	10	LF	
0B	013	11	VT	
0C	014	12	FF	
0D	015	13	CR	
0E	016	14	SO	
0F	017	15	SI	
10	020	16	DLE	
11	021	17	DC1	LLO
12	022	18	DC2	
13	023	19	DC3	
14	024	20	DC4	DCL
15	025	21	NAK	PPU
16	026	22	SYN	
17	027	23	ETB	
18	030	24	CAN	SPE
19	031	25	EM	SPD
1A	032	26	SUB	
1B	033	27	ESC	
1C	034	28	FS	
1D	035	29	GS	
1E	036	30	RS	
1F	037	31	US	

20	040	32	SP	MLA0
21	041	33	!	MLA1
22	042	34	"	MLA2
23	043	35	#	MLA3
24	044	36	$	MLA4
25	045	37	%	MLA5
26	046	38	&	MLA6
27	047	39	'	MLA7
28	050	40	(	MLA8
29	051	41	)	MLA9
2A	052	42	*	MLA10
2B	053	43	+	MLA11
2C	054	44	,	MLA12
2D	055	45	-	MLA13
2E	056	46	.	MLA14
2F	057	47	/	MLA15
30	060	48	0	MLA16
31	061	49	1	MLA17
32	062	50	2	MLA18
33	063	51	3	MLA19
34	064	52	4	MLA20
35	065	53	5	MLA21
36	066	54	6	MLA22
37	067	55	7	MLA23
38	070	56	8	MLA24
39	071	57	9	MLA25
3A	072	58	:	MLA26
3B	073	59	;	MLA27

3C	074	60	<	MLA28
3D	075	61	=	MLA29
3E	076	62	>	MLA30
3F	077	63	?	UNL
40	100	64	@	MTA0
41	101	65	A	MTA1
42	102	66	B	MTA2
43	103	67	C	MTA3
44	104	68	D	MTA4
45	105	69	E	MTA5
46	106	70	F	MTA6
47	107	71	G	MTA7
48	110	72	H	MTA8
49	111	73	I	MTA9
4A	112	74	J	MTA10
4B	113	75	K	MTA11
4C	114	76	L	MTA12
4D	115	77	M	MTA13
4E	116	78	N	MTA14
4F	117	79	O	MTA15
50	120	80	P	MTA16
51	121	81	Q	MTA17
52	122	82	R	MTA18
53	123	83	S	MTA19
54	124	84	T	MTA20
55	125	85	U	MTA21
56	126	86	V	MTA22
57	127	87	W	MTA23

58	130	88	X	MTA24
59	131	89	Y	MTA25
5A	132	90	Z	MTA26
5B	133	91	[	MTA27
5C	134	92	\	MTA28
5D	135	93	]	MTA29
5E	136	94	^	MTA30
5F	137	95	_	UNT
60	140	96	`	MSA0,PPE
61	141	97	a	MSA1,PPE
62	142	98	b	MSA2,PPE
63	143	99	c	MSA3,PPE
64	144	100	d	MSA4,PPE
65	145	101	e	MSA5,PPE
66	146	102	f	MSA6,PPE
67	147	103	g	MSA7,PPE
68	150	104	h	MSA8,PPE
69	151	105	i	MSA9,PPE
6A	152	106	j	MSA10,PPE
6B	153	107	k	MSA11,PPE
6C	154	108	l	MSA12,PPE
6D	155	109	m	MSA13,PPE
6E	156	110	n	MSA14,PPE
6F	157	111	o	MSA15,PPE
70	160	112	p	MSA16,PPD
71	161	113	q	MSA17,PPD
72	162	114	r	MSA18,PPD
73	163	115	s	MSA19,PPD

74	164	116	t	MSA20,PPD
75	165	117	u	MSA21,PPD
76	166	118	v	MSA22,PPD
77	167	119	w	MSA23,PPD
78	170	120	x	MSA24,PPD
79	171	121	y	MSA25,PPD
7A	172	122	z	MSA26,PPD
7B	173	123	{	MSA27,PPD
7C	174	124	|	MSA28,PPD
7D	175	125	}	MSA29,PPD
7E	176	126	~	MSA30,PPD
7F	177	127	DEL	

Message Definitions

DCL	Device Clear
GET	Group Execute Trigger
GTL	Go To Local
LLO	Local Lockout
MLA	My Listen Address
MSA	My Secondary Address
MTA	My Talk Address
PPC	Parallel Poll Configure
PPD	Parallel Poll Disable
PPE	Parallel Poll Enable
PPU	Parallel Poll Unconfigure
SDC	Selected Device Clear
SPD	Serial Poll Disable
SPE	Serial Poll Enable

TCT	Take Control
UNL	Unlisten
UNT	Untalk

Troubleshooting

If you cannot communicate with the GPIB-RS232/485/422 or your instrument, the following suggestions may help troubleshoot your problem.

- If the **PWR** LED is not lit, verify that the power cable is connected.
- If the **RDY** LED is steady red and the **BUSY** LED is steady yellow, the firmware image inside the GPIB-RS232/485/422 has been corrupted. Most likely, this happened as a result of disconnecting power during the previous firmware update. To recover from this condition, rerun the NI GPIB-Serial Converter Wizard and follow the instructions described in Using the NI GPIB-Serial Converter Wizard to Update the GPIB-RS232/485/422 Firmware.
- If the **RDY** LED is steady red, contact National Instruments for support.
- If the **RDY** LED is flashing green, the box is powered on in configuration mode. To configure your box, refer to Using the NI GPIB-Serial Converter Wizard.
- If the **RDY** LED is a steady green, you are booted into one of four converter box modes. Refer to the S Mode, G Mode, C Mode, and D Mode documentation for more information about each mode. If you do not know what mode your box is in, refer to Using the NI GPIB-Serial Converter Wizard.
- If you are using S or C Mode, ensure that you have configured all serial port settings on the GPIB-RS232/485/422 to match the host machine's settings.
- If you are using C mode, ensure that you have configured the correct Primary and Secondary GPIB address of the device you are talking to.
- If you are using G or D Mode, pay close attention to the Primary and Secondary address you have configured the GPIB-RS232/485/422 to use. Be sure your program is using the correct GPIB addresses.
- In all modes, if you are using hardware handshaking, verify that your cable and serial device support it. Refer to Hardware Handshaking and the flow control topic for the mode you are using for more details about serial flow control behavior.

Differences Between the GPIB-232/485CT/CV-A and GPIB-

RS232/485/422

S Mode and G Mode

Enabling/Disabling XON/XOFF Flow Control

With the GPIB-232/485CT-A, you could enable the XON/XOFF protocol when sending data to the serial port independently from enabling the protocol when receiving data from the serial port.

The GPIB-RS232/485/422 does not allow such flexibility. It allows you to either enable the XON/XOFF protocol when sending and receiving data to/from the serial port, or disable the XON/XOFF protocol altogether. Thus, `xon 0,1` and `xon 1,0` are still syntactically valid, but they have the side effect of enabling the XON/XOFF protocol when the GPIB-RS232/485/422 both sends and receives serial data.

Function Arguments Syntax

The GPIB-RS232/485/422 enforces the following syntax rules for the function arguments: The first argument is separated from the function name with at least once space, and each additional argument is separated with at least once space or comma. The GPIB-232/485/CT/CV-A allowed no separation between function arguments.

id string

The id string has changed between the GPIB-232/485/CT/CV-A and GPIB-RS232/485/422.

The GPIB-232/485/CT/CV-A general id string format was as follows:

```
GPIB-232/485CT-A Rev. B.3<CR><LF>
(c)1995 National Instruments<CR><LF>
256K bytes RAM<CR><LF>
```

The GPIB-RS232/485/422 general id string format is either of the following two formats:

```
GPIB-RS232<CR><LF>
(c)2006 National Instruments<CR><LF>
Version 1.0.0f0<CR><LF>

GPIB-RS485 / RS422<CR><LF>
(c)2006 National Instruments<CR><LF>
Version 1.0.0f0<CR><LF>
```

S Mode

tmo Timeout Value Range

The GPIB-232/485/CT/CV-A timeout value range for both I/O and serial poll was between 0.00001 and 3600 seconds. The GPIB-RS232/485/422 accepts timeout values in the same range as valid parameter values. However, the effective timeout range allowed is between 0.0001 and 1000 seconds. Thus, any value less then 0.0001 seconds is treated as 0.0001 seconds, and any value greater then 1000 seconds is treated as 1000 seconds.

tmo Query Drops Trailing Zeros

When issued a `tmo` query, the GPIB-232/485/CT/CV-A returned values exactly as it was issued in the last `tmo <ioval>, <spval>` command. For example, if the `tmo 10.0300` command was issued, the GPIB-232/485/CT/CV-A returned `10.0300`. The GPIB-RS232/485/422 returns `10.03`. Also, the GPIB-RS232/485/422 timeout default value is `10,0.1`, whereas the GPIB-232/485/CT/CV-A default was `10.0,0.1`.

echo Command Echoes Every Character

If <CR><LF> was used to terminate commands sent to the GPIB-232/485/CT/CV-A, it echoed only <CR>, but not <LF>. The GPIB-RS232/485/422 echoes both <CR> and <LF>.

sic

With the GPIB-232/485CT-A, you could provide a time argument to the `sic` command to change the amount of time the IFC line is asserted. The GPIB-RS232/485/422 ignores the time argument, but does not report an error. The GPIB-RS232/485/422 asserts IFC for at least 100 µs.

onl 1 and Restoring All Settings to Their Power-On Values

When the `onl 1` command is issued to the GPIB-RS232/485/422, it restores all settings to their power-on values except the ones changed with the `xon` command. For example, if you issued the `xon 1,1` command to enable XON/XOFF handshaking, followed by the `onl 1` command, XON/XOFF handshaking is still enabled. The GPIB-232/485/CT/CV-A reset all settings to power-on values, including the ones changed with the `xon` command.

G Mode

GPIB-RS232/485/422 Device and <CR><LF>

The GPIB-RS232/485/422 does not return <CR><LF> when no response is available (that is, no query was sent and continuous status is not enabled). If you attempt to read from the GPIB-RS232/485/422, the read times out.

Serial Polls

The GPIB-232/485/CT/CV-A responded to serial polls on either the GPIB-232/485/CT/CV-A or serial device address. If the GPIB-232/485/CT/CV-A was asserting SRQ*, serial polling either address would make the GPIB-232/485/CT/CV-A unassert SRQ*. The GPIB-RS232/485/422 responds only to serial polls on the GPIB-RS232/485/422 address. Also, if the GPIB-RS232/485/422 is asserting SRQ*, serial polling only the GPIB-RS232/485/422 address makes the GPIB-RS232/485/422 unassert SRQ*.

Parallel Polls

The GPIB-RS232/485/422 does not support parallel poll functionality at this time. The GPIB-232/485/CT/CV-A implemented IEEE 488 Parallel Poll (PP) interface function subset PP1.

D Mode

Serial Buffer Size is Not Configurable

The GPIB-232CV-A had 256 KB of memory, shared between GPIB and serial buffers. The user had an option of setting the serial buffer size to be 32 bytes or 32 KB, with the

GPIB buffer taking the remainder. The GPIB-RS232/485/422 has a 1 MB GPIB buffer and 1 MB serial buffer. Both buffers are significantly large to accommodate most applications, so a design decision was made to disallow the user to choose buffer sizes.

C Mode

Configuring No Address

The GPIB-232CV-A has an option to set all SW1 switches 1 through 5 on. The GPIB-RS232/485/422 does not have this option.

Terminating with EOI

On the GPIB-RS232/485/422, both <CR> and <LF> termination modes have the configurable option of asserting EOI (End Or Identity) on the GPIB bus on the terminating byte prior to readdressing the device as a Talker.

All Modes

The GPIB-RS232/485/422 now supports secondary addressing.

Hardware Handshaking

The CTS line default state (when not driven by a serial device on the other side) on the GPIB-232/485/CT/CV-A was pulled low. However, the CTS line default state on the GPIB-RS232/485/422 is pulled high. This means that if you enable hardware handshaking on the GPIB-RS232/485/422, and your serial cable or serial device does not support it, or if you did not properly connect RTS and CTS signals as described in Hardware Handshaking, the GPIB-RS232/485/422 cannot send serial data to your serial device.

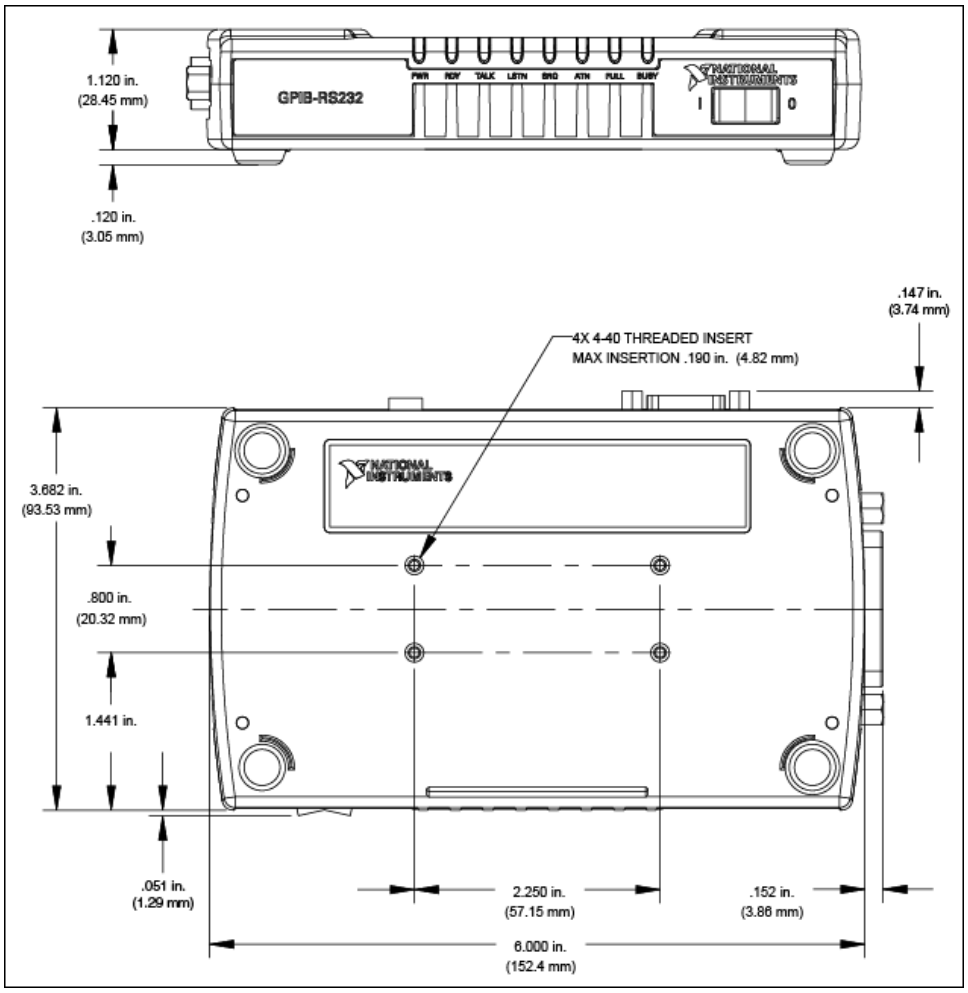
Do not enable hardware handshaking on the GPIB-RS232/485/422 if you suspect the conditions described above are true for your system.

Refer to Hardware Handshaking and the flow control topic for the mode you are using for more details about serial flow control behavior for each mode.

BUSY LED

On the GPIB-232/485/CT/CV-A, this LED indicates that the device is currently accepting serial data. Each serial character received toggles the **BUSY** LED status. The behavior of this LED in the GPIB-RS232/485/422 is described in LEDs.

Physical Dimensions



Glossary

Prefixes | Numbers/Symbols | A | C | D | E | F | G | H | I | L | M | O | P | R | S | T | U | V

Prefixes

Symbol	Prefix	Value
p	pico	10^{-12}
n	nano	10^{-9}
µ	micro	10^{-6}
m	milli	10^{-3}
k	kilo	10^{3}
M	mega	10^{6}
G	giga	10^{9}
T	tera	10^{12}

Numbers/Symbols

nV	nanovolts	10^{-9} volts
µV	microvolts	10^{-6} volts
µΩ	microohms	10^{-6} ohms
mΩ	milliohms	10^{-3} ohms
MΩ	megaohms	10^{6} ohms
pA	picoamps	10^{-12} amperes
nA	nanoamps	10^{-9} amperes
µA	microamps	10^{-6} amperes
mA	milliamps	10^{-3} amperes

A

A	amperes

AC	alternating current
ANSI	American National Standards Institute
ASCII	American Standard Code for Information Interchange
ATN	Attention
C	
C	Celsius
CIC	Controller-In-Charge
CSA	Canadian Standards Association
CTS	Clear to Send
D	
DCAS	Device Clear Active State
DCE	Data Circuit-terminating Equipment
DCL	Device Clear
DIO	digital input/output
DIP	dual inline package
DTAS	Device Trigger Active State
DTE	Data Terminal Equipment
DTR	Data Terminal Ready
E	
EIA	Electronic Industries Association
EOI	end or identify
EOS	end of string
F	
FCC	Federal Communications Commission

G	
g	grams
GND	ground
GPIB	General Purpose Interface Bus
H	
hex	hexadecimal
Hz	hertz
I	
IDY	Identify
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronic Engineers
IFC	Interface Clear
in.	inches
I/O	input/output
ISO	International Standards Organization
IST	Individual Status
L	
LACS	Listener Active State
LED	light-emitting diode
LSI	large scale integration
M	
MB	megabytes of memory
m	meters

O	
oz	ounces
P	
PC	personal computer
PPE	Parallel Poll Enable
R	
RAM	random-access memory
REM	Remote
REN	Remote Enable
ROM	read-only memory
RQS	Request Service
RTS	Request to Send
RXD	Receive Data
S	
s	seconds
SRQ	Service Request
SRQI	Service Request Input
T	
TACS	Talker Active State
TIMO	Timeout
TTL	Transistor-Transistor Logic
TXD	Transmit Data

U	
UL	Underwriters Laboratories
V	
V	volts
VAC	volts alternating current
VDC	volts direct current

Important Information

Warranty

Copyright

Trademarks

Patents

Warning Regarding Use of NI Products

Warranty

The GPIB-RS232/485/422 is warranted against defects in materials and workmanship for a period of two years from the date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace equipment that proves to be defective during the warranty period. This warranty includes parts and labor.

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 document 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

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

National Instruments respects the intellectual property of others, and we ask our users to do the same. NI software is protected by copyright and other intellectual property laws. Where NI software may be used to reproduce software or other materials belonging to others, you may use NI software only to reproduce materials that you may reproduce in accordance with the terms of any applicable license or other legal restriction.

Trademarks

National Instruments, NI, `ni.com`, and LabVIEW are trademarks of National Instruments Corporation. Refer to the ***Terms of Use*** section on `ni.com/legal` for more information about National Instruments trademarks.

FireWire® is the registered trademark of Apple Computer, Inc.

Handle Graphics®, MATLAB®, Real-Time Workshop®, Simulink®, Stateflow®, and xPC TargetBox® are registered trademarks, and TargetBox™ and Target Language Compiler™ are trademarks of The MathWorks, Inc.

Tektronix® and Tek are registered trademarks of Tektronix, Inc.

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

Members of the National Instruments Alliance Partner Program are business entities independent from National Instruments and have no agency, partnership, or joint-venture relationship with National Instruments.

Patents

For patents covering National Instruments products, refer to the appropriate location: **Help»Patents** in your software, the `patents.txt` file on your CD, or ni.com/patents.

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

(1) NATIONAL INSTRUMENTS PRODUCTS ARE NOT DESIGNED WITH COMPONENTS AND TESTING FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS WHOSE FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO CAUSE SIGNIFICANT INJURY TO A HUMAN.

(2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE FITNESS, FITNESS OF COMPILERS AND DEVELOPMENT SOFTWARE USED TO DEVELOP AN APPLICATION, INSTALLATION ERRORS, SOFTWARE AND HARDWARE COMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATIONS DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED "SYSTEM FAILURES"). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE RELIANT SOLELY UPON ONE FORM OF ELECTRONIC SYSTEM DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE, INJURY, OR DEATH, THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES, INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS. BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFERS FROM NATIONAL INSTRUMENTS' TESTING PLATFORMS AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE NATIONAL INSTRUMENTS PRODUCTS IN COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY NATIONAL INSTRUMENTS, THE USER OR APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF NATIONAL INSTRUMENTS PRODUCTS WHENEVER NATIONAL INSTRUMENTS PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

Technical Support and Professional Services

Visit the following sections of the National Instruments Web site at ni.com for technical support and professional services:

- Support—Online technical support resources at ni.com/support include the following:
  - **Self-Help Resources**—For answers and solutions, visit the award-winning National Instruments Web site for software drivers and updates, a searchable KnowledgeBase, product manuals, step-by-step troubleshooting wizards, thousands of example programs, tutorials, application notes, instrument drivers, and so on.
  - **Free Technical Support**—All registered users receive free Basic Service, which includes access to hundreds of Applications Engineers worldwide in the NI Developer Exchange at ni.com/exchange. National Instruments Applications Engineers make sure every question receives an answer.

    For information about other technical support options in your area, visit ni.com/services or contact your local office at ni.com/contact.

- Training and Certification—Visit ni.com/training for self-paced training, eLearning virtual classrooms, interactive CDs, and Certification program information. You also can register for instructor-led, hands-on courses at locations around the world.
- System Integration—If you have time constraints, limited in-house technical resources, or other project challenges, National Instruments Alliance Partner members can help. To learn more, call your local NI office or visit ni.com/alliance.

If you searched ni.com and could not find the answers you need, contact your local office or NI corporate headquarters. You also can visit the Worldwide Offices section of ni.com/niglobal to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

© 2024 National Instruments Corporation.

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP