
# Applications Note

## GPIB Communication with a Pegasus™ Prober

Wentworth Laboratories

# GPIB Communication with a Pegasus™ Prober

*IEEE-488 refers to the Institute of Electrical and Electronics Engineers (IEEE) Standard number 488 and is commonly called GPIB (General Purpose Interface Bus). The GPIB interface can be used to transfer data between two or more devices. It is particularly well suited for interconnecting computers and instruments.*

This document briefly describes how to control a Wentworth Pegasus™ S200/S300 series prober using a National Instruments (NI) IEEE 488.2 (GPIB) card or USB adapter.

The Command Protocol section describes the GPIB protocol used by the prober. The *'Using the NI VISA Test Panel, Using NI Interactive Control (with 488.1 Functions), Using NI Interactive Control (with 488.2 Functions)'* sections describe how to send a command using various NI tools. The example C/C++ Application (with 488.1 Functions) section describes how to send a command from a program.

Though out this document the phrase 'prober' will be used to mean Pegasus S200/300 series prober. The IEEE 488.2 bus will be referred to by its alternative name of the GPIB, and the IEEE 488.1/2 standards will be abbreviated to 488.1/2.

- This document is intended to supplement the Pegasus prober's user manual, and should be read with that manual. A copy of this manual can be found on the Manuals & Utilities USB shipped with every prober.
- This document assumes a basic knowledge of the GPIB bus and the NI Measurement & Automation Explorer (MAX), and should be read with NI GPIB Manuals.

## Contents

# 1. Command Protocol

The command protocol of the prober is a master/slave protocol with the controlling device acting as the master, and the prober acting as the slave. The protocol itself is string based, and can be best described as a Write/Wait for SRQ/Read protocol.

The normal command sequence is as follows:

- The master writes a line-feed terminated command string to the prober.
- The master waits for the prober to send a service request.
- The master acknowledges the service request and reads the serial poll response byte.
- The masters read the line-feed terminated reply string from the prober.

The prober flags an error by setting the top bit of the serial poll response byte.
If this happens, the reply will always contain an INF code (see the prober's user manual for details).

If no error is flagged, the reply will either contain the reply to the command or an `INF 000` if the command doesn't return a reply (this is command dependent, see the prober's user manual for detail of the individual commands and their replies).

- All command strings must be terminated by a line-feed character (\n).
- All reply strings will be terminated by a line-feed character (\n).
- The master must always read the reply string if a service request is received.

# 2. Using the NI VISA Test Panel

The prober's command protocol is not compatible with the NI 488.2 Communicator tool. The NI VISA Test Panel tool must be used instead of the NI 488.2 Communicator.

The **NI VISA Test Panel** is a graphical tool that allows low level communication with a device.

To run **NI VISA Test Panel** tool, simply use the following steps:

- Run the NI Measurement & Automation Explorer (MAX)
- Expand the Devices and Interfaces item
- Right click on **GPIB0**, and then select **Open VISA Test Panel**

To send a command to the prober, simply enter the following lines (this example uses the GID command, but any command could be used):

- Select the **Input/Output** tab
- Type the command into the **Select or Enter Command** input box (e.g., `GID\n`)
- Press the **Write** button
- Press the **Read Status Byte** button, until the status byte stars with `4`, `5`, `6`, `7`, `C`, `D`, `E`, or `F`.
- Press the **Read** button

# 3. Using NI Interactive Control (with 488.1 Functions)

The *'Using NI Interactive Control (with 488.1 Functions)'* and *'Using NI Interactive Control (with 488.2 Functions)'* sections intentionally duplicate information, so the reader can read the relevant section and skip the other one.

If you are using this section to troubleshoot a GPIB problem, we advise you to unplug any other device from the GPIB bus before following these steps.

The **NI Interactive Control** tool is a command line interface that allows individual 488.1 functions to the called. This is useful for familiarization with the GPIB interface and for testing a GPIB connection.

To run **NI Interactive Control** tool, simply use the following steps:

- Run the NI Measurement & Automation Explorer (MAX)
- Expand the Devices and Interfaces item
- Right click on **GPIB0**, and then select **Interactive Control**

To start communication with the prober using this tool, simply enter the following lines:

- ibfind “DEV3”
- ibrsp
- ibrd 99

The *ibfind* line opens the device at address 3 on the GPIB bus (the default address for the **Pegasus** prober).

The *ibrsp* line checks for an initial response from the prober. If bit 6 of the serial poll response byte is set, then the *ibrd* line must be used to read the reply before continuing. If bit is not set, then the *ibrd* line is not needed (and will return a read timeout error).

To send a command to the prober, simply enter the following lines (this example uses the LDI command, but any command could be used):

- ibwrt "LDI\n"
- ibrsp
- ibrd 99

The *ibwrt* line writes the LDI (reference all motors) command to the prober, which must be line-feed terminated.

The *ibrsp* line should be entered after the prober has finished referencing, and will show a service request by having bit 6 of the serial poll response byte set.

The *ibrd* line reads the reply from the LDI command, which is line-feed terminated.

Alternatively, the *ibwait* function (with a suitable timeout value set by the *ibtmo* function) could be used instead of polling using the *ibrsp* function.

# 4. Using NI Interactive Control (with 488.2 Functions)

The *'Using NI Interactive Control (with 488.1 Functions)'* and *'Using NI Interactive Control (with 488.2 Functions)'* sections intentionally duplicate information, so the reader can read the relevant section and skip the other one.

If you are using this section to troubleshoot a GPIB problem, we advise you to unplug any other device from the GPIB bus before following these steps.

The **NI Interactive Control** tool is a command line interface that allows individual 488.2 functions to be called. This is useful for familiarization with the GPIB interface and for testing a GPIB connection.

To run **NI Interactive Control** tool, simply use the following steps:

- Run the NI Measurement & Automation Explorer (MAX)
- Expand the Devices and Interfaces item
- Right click on **GPIB0**, and then select **Interactive Control**

To start communication with the prober using this tool, simply enter the following lines:

- set 488.2
- SendIFC
- TestSRQ
- ReadStatusByte 3
- Receive 3 99 STOPend

The *set* line is not a 488.2 function, it just changes the tool from 488.1 to 488.2 mode.

The *SendIFC* line opens the GPIB bus.

The *TestSRQ* line checks to see if a service request has been received from any device. If one has, then the *ReadStatusByte* line reads the status byte of the prober at address DEV3 (the default address for the prober). If bit 6 of the status byte is set, then the *Receive* line must be used to read the reply before continuing. If no service request has been received, then the *ReadStatusByte* and *Receive* lines are not needed (and the *Receive* line will return a read timeout error). If the service request is not from the prober, then the *Receive* line is not needed (and will timeout).

To send a command to the prober, simply enter the following lines (this example uses the *LDI* command, but any command could be used):

- Send 3 "LDI\n" DABend
- TestSRQ
- ReadStatusByte 3
- Receive 3 99 STOPend

The *Send* line writes the *LDI* (reference all motors) command to the prober, which must be line-feed terminated.

The *TestSRQ* line should be entered after the prober has finished referencing.

The *ReadStatusByte* line reads the status byte and the *Receive* line reads the reply from the *LDI* command, which is line-feed terminated.

Alternatively, the *WaitSRQ* function (with a suitable timeout value set by the *ibconfig* function) could be used instead of polling using the *TestSRQ* function.

# 5. Example C/C++ Application (with 488.1 Functions)

The following sample C/C++ code demonstrates how to send an Initial Load command to a prober using 488.1 functions. The sample code is written in C/C++, but the principles apply to all programming languages supported by NI (including C++, C#, Visual Basic, and Delphi).

The command sequence is the same as in the Interactive Communication section, except that the *ibwait* command is used to wait for the Initial Load (*LDI)* command to finish.

If background processing is required while waiting for a command to finish, the *ibrsp* command can be used to periodically poll the prober.

```
//---------------------------------------------------------------------
#include <stdio.h> // For sprintf()
#define STRICT
#include <windows.h>
#include "decl-32.h"

static int ProberGPIB_Open(const char *szDevice);
static int ProberGPIB_SendCommand(int iDevice,
        const char *szCommand,
        char *pcResponseByte,
        char *szReply,
        int iReplySize);
//---------------------------------------------------------------------

#pragma argsused
int WINAPI WinMain(HINSTANCE hInstance,
        HINSTANCE hPrevInstance,
        LPSTR lpCmdLine,
        int nCmdShow)
{
        char szText[200];
        int iDevice;
        int iStatus;
        char cResponseByte;
        char szReply[100];

        iDevice = ProberGPIB_Open("DEV3");
        sprintf(szText, "Device descriptor = %d", iDevice);
        MessageBox(NULL, szText, "ProberGPIB_Open", MB_TASKMODAL);
        iStatus = ProberGPIB_SendCommand(iDevice, "LDI\n",
                &cResponseByte,
                szReply, sizeof(szReply));
        if (iStatus==0)
        {
                sprintf(szText, "Reply = %s\nSerial poll response byte = %02X",
                        szReply, (int)cResponseByte);
        }
        else
        {
                sprintf(szText, "GPIB fault: ibsta = %04X", iStatus);
```

```
	}
	MessageBox(NULL, szText, "ProberGPIB_SendCommand", MB_TASKMODAL);

	return (0);
}
//---------------------------------------------------------------------

static int ProberGPIB_Open(const char *szDevice)
{
	int iDevice;
	unsigned char cResponseByte;
	char szInitString[20];

	iDevice = ibfind(szDevice);
	// Read INF120 power-on string (falls through if already read)
	cResponseByte = 0xBF;
	while (cResponseByte==0xBF)
	{
		ibrsp(iDevice, (char *)&cResponseByte);
	}
	if ((cResponseByte&0x40)!=0)
	{
		ibrd(iDevice, szInitString, sizeof(szInitString));
	}

	return (iDevice);
}
//---------------------------------------------------------------------

static int ProberGPIB_SendCommand(int iDevice,
	const char *szCommand,
	char *pcResponseByte,
	char *szReply,
	int iReplySize)
{
	int iStatus;

	ibwrt(iDevice, (char *)szCommand, strlen(szCommand));
	iStatus = ibwait(iDevice, TIMO|RQS); // Wait for SRQ or timeout
	if ((iStatus&RQS)!=0)
	{
		ibrsp(iDevice, pcResponseByte); // Acknowledge SRQ
		ibrd(iDevice, szReply, iReplySize);
		szReply[ibcntl] = '\0'; // Nul terminate the string!
		iStatus = 0;
	}

	return (iStatus);
}
//---------------------------------------------------------------------
```