Using Measurement Studio GPIB to Accelerate Development with Visual Basic

Jason White and Evan Cone

Introduction

Using GPIB in Visual Basic can be a complicated experience. One of the greatest problems when getting started with GPIB is verifying that you can communicate with your GPIB instruments. You might spend hours sifting through lengthy manuals and layers of menus on the instrument trying to find information about the GPIB address and whether GPIB is enabled. Sometimes trying to get the GPIB instrument to respond to an identification query can be difficult.

Once you communicate with your instrument, the next step is to develop an application that can make full use of virtual instrumentation – the combination of your PC and instruments. With the Measurement Studio GPIB (CWGPIB) ActiveX control, you can simplify and speed up the development process. This application note explains how to quickly determine the GPIB address, communicate with your instrument, and configure it at design time.

Note Measurement Studio includes tools to build measurement applications in ANSI C, Visual C++, and Visual Basic. The Measurement Studio Visual Basic tools were formerly known as ComponentWorks.

Getting Started – Fast and Easy

The CWGPIB control contains many features that make getting started with GPIB fast and easy. Using the property pages of the control, you can interactively find and communicate with your GPIB instruments at design time. With the CWGPIB control, you can speed up application development by learning how to automate measurements with your instruments, uncover GPIB problems, and identify malfunctioning instruments. Because you can use the convenient debugging tools of the control at design time, you avoid repeatedly recompiling your application to verify communication with your instrument.

Note For more information about GPIB and setting up your controller, refer to the GPIB user and function reference manuals you received with your GPIB controller. This application note assumes that you have a basic knowledge of GPIB.

Quickly Determining the GPIB Address

1. Launch Visual Basic from the Windows Start menu.
2. Open a new Standard EXE project.
3. Right click on the Visual Basic Toolbox and select Components.
5. Place a CWGPIB control on the form or dialog box.
Right click on the control and select **Properties** to bring up the **General** property page, as shown in Figure 1. The **Devices** combo box contains a list of the listening devices active on the bus. To quickly locate a GPIB address, select the device that you want to communicate with from the combo box. If your device is not automatically detected, you can manually specify the address from **Manual Select** section.

**Tip** If you have National Instruments hardware installed, you can use the Measurement & Automation Explorer to determine the board number, primary address, and secondary address.

![Figure 1. Selecting a GPIB Instrument from the General Property Page](image)

If you have two or more instruments on the bus and you are not sure which address corresponds to each instrument, disconnect all but one instrument and click **Update** to refresh the **Devices** list. Repeat this task for each instrument. You also can query each instrument for its identification (ID) string to match each instrument with its address.

### Establishing Communication with Your Instruments

Once you have determined the GPIB address of your instruments, you must verify that you can send and receive data to and from the instrument. Because most instruments are 488.2 compliant, you can query the instrument for its identification by sending it the "*IDN?" command. Instruments typically respond with the manufacturer name, model name, and various alphanumeric characters that the manufacturer uses to track firmware revisions. You can verify communication with an instrument in the CW GPIB control property pages:

1. Open the **General** property page and select the instrument from the **Devices** list.
2. Open the **Test** property page and enter "*IDN?" in the **Send string** text box (the command appears there by default).
3. Click the **Execute** button.

The response from the instrument is displayed in the **String received** panel. The result from a Fluke 45 digital multimeter is shown in Figure 2.
Figure 2. Interactive Communication with a Fluke 45 Multimeter Using the Test Property Page

Note: If you have a 488.1 instrument, consult its documentation to find an appropriate command that provides data that can be read back from the instrument. 488.1 devices do not typically respond to a \*IDN? command.

The Execute button does both a write and a read operation on the instrument. You also can use the Write and Read buttons to do individual write and read operations. For example, if you want to send multiple write commands to an instrument before doing a read operation, use the Write and Read buttons.

Errors and status from the operations are displayed in the Globals and Status areas on the Test property page. Refer to the GPIB reference included with your controller for more information about the values of these status variables.

Design-Time Configuration

You can set more advanced GPIB configurations, such as Timeout, Unaddressing mode, and EOS mode from the Advanced property page. You can then go back to the Test property page and check the result of changing these settings.

For an experienced GPIB user, these basic functions and the concepts of verifying a GPIB address and instrument communication might seem simplistic. However, these getting-started tools are invaluable in troubleshooting and helping you focus on developing your test applications.

Productivity Tools

Once you have established communication with your instruments and ensured that the GPIB system is working properly, you are ready to write your tests. The CWGPIB control provides several productivity tools to accelerate development and make writing tests easier, as well as some advanced features to make your applications faster and more powerful. The control includes a parsing tool with which you can graphically or programmatically specify the format in which you want the data returned. You also can use asynchronous notification of GPIB events.
**Parsing Has Never Been Easier**

Instruments are notorious for sending back cryptic binary strings or strings containing a combination of header and data information. Often, it can often be frustrating trying to make your application work with these strings. Parsing them into a usable data format can be a time-consuming process. The CWGPIB control provides built-in data parsing to make working with instruments easier. You can use this interactive string parsing tool to define rules for parsing information out of instrument strings so you keep only the data you need.

Using the Parsing property page of the control, you specify a parsing task (a way to parse the data). You can then execute this task from your code to automatically parse the data returned from an instrument. You can specify as many parsing tasks as you need, so that each type of interaction with an instrument can be parsed differently. The control also provides a built-in task called the **Number Parser**, which converts a comma-delimited (or any other delimited) string into an array of numbers, stripping out headers and other nonnumerical information.

Let's take a look at a simple example to show the power and ease of use of the CWGPIB control. Suppose that we have an instrument that responds to the "DATA?" command by returning a comma-separated list of numbers preceded with some header information. The string might look something like "WFM 3.246,3.295,4.653,3.334". Typically, you must make several function calls to configure the communication parameters, such as addressing and EOS characters. After making these configuration calls, you must write the "DATA?" command to the instrument. Finally, you read the response back from the instrument. You receive a data string that is in the same format shown above. You must then go through the string and parse it to pull out each of the ASCII represented numbers and convert them to a real numeric type. Without the CWGPIB control, you must use the following Visual Basic code:

```vbnet
'------ GPIB communication using dll functions ------
Dim ud as Integer
Dim sendstr As String, s As String
sendstr = "DATA?"
Call ibdev(0, 2, 0, 10, 1, 0, ud)   configure
If iberr Then HandleError   'Error handler code (not shown here)
Call ibwrt(ud, sendstr)      'Write "DATA?"
If iberr Then HandleError   'Error handler code (not shown here)
s = Space(2000)
Call ibrd(ud, s)             'Read
If iberr Then HandleError   'Error handler code (not shown here)

' ------ parse data ------
Dim r(128) As Double         'r is the array of doubles
'Assumes known amount of data (don’t have to resize)
'The CWGPIB control handles resizing automatically
Dim i, j, k, length As Integer

i = 1                        'i is the current character
k = 1                        'k is the index of the array of values
length = Len(s)              
```

Note For brevity, certain assumptions were made in this code. The data always returns 128 values. If it returns an unknown amount of data (specified in the header, perhaps) then resizing of the data array must be done inside the code. The CWGPIB control handles resizing and returns the correctly sized array. Also, the error handler code is not shown here.
'Strip out header (assumes header is followed by space)
Do While (Mid(s, i, 1) <> " ") and i < length
    i = i + 1
Loop

j = i                        'j is the index of the beginning of a new number
'Continue going through string until the end
Do While i < length
    'Iterate through string until the current character is a comma
    Do While (Mid(s, i, 1) <> ",") And i < length
        i = i + 1
    Loop
    'Add number to array and update counters
    'NOTE: resizing of the array would have to be done here for variable size data
    r(k) = Val(Mid(s, j, i - j + 1))
    k = k + 1
    i = i + 1
    j = i
Loop

This simple example can turn into a lot of code if you have to manually take care of configuration and data parsing. Now, look at the same code using the CWGPIB control:

Dim data As Variant                 'The Variant type will be an array of doubles
CWGPIB1.Configure                   'Perform all configuration as specified by the
                                     'property pages
CWGPIB1.Write "DATA?"               'Writes the "DATA?" command to the instrument
data = CWGPIB1.Tasks.Item("Number Parser").Read
                                     'Reads the response from the instrument, and runs
                                     'The number parser task on the data before
                                     'returning it to the data array. Data will thus
                                     'be an array containing:
                                     'data(0) = 3.246
                                     'data(1) = 3.295
                                     'data(2) = 4.653
                                     'data(3) = 3.334

You can then directly manipulate or examine the waveform, write it out to a database or spreadsheet, or pass it directly to a Measurement Studio Graph ActiveX control for plotting of the waveform.

Note  For more information about parsing instrument data using the Measurement Studio ActiveX controls, refer to National Instruments Application Note 148, *Parsing Instrument Data in Visual Basic*, at ni.com

Asynchronous Notification of Events

While traditional GPIB programming relies on waiting for synchronous commands to end, the Measurement Studio CWGPIB control handles asynchronous operations, which includes event-driven programming. You can write code that executes whenever certain conditions on the GPIB bus are met – the control handles all of the asynchronous I/O for you. You can optimize your programs to run faster and more smoothly. Refer to National Instruments Application Note 100, *How to Use Asynchronous Callback Functions with GPIB Events in Windows NT/98/95*, at ni.com
Building Complete Solutions Faster

Another frustrating aspect of building complete solutions is repeating steps. For example, you configure your application to work with an instrument and then in your next application, you must redo all of those configuration steps again. Measurement Studio provides several features to help you build complete system solutions faster.

Reusable Instrument-Specific GPIB Controls

The CWGPIB control saves you time because you can import control styles so that you configure your control only once for a particular instrument. You can then export that control style and compile a library of instrument-specific controls, so you can reuse your configurations across other applications.

Once you have finished configuring the settings on the CWGPIB control to work with a certain instrument, you can then save those settings by right clicking on the control and selecting Export Style. When writing other applications with that instrument, you can then import this style by right clicking on the control and selecting Import Style. You also can import those settings at run time for dynamic instrument communication.

Instrument Drivers

To simplify your instrument control tasks further, Measurement Studio is shipped with numerous instrument drivers. You can choose not to install the instrument drivers if they are already installed in the system. Instrument drivers are powerful pieces of software that combine the granular, low-level command strings needed for configuration and measurement into higher-level functions, such as initialize, reset, and read waveform. Instrument libraries make instrument control easier and help you focus on taking measurements, developing complete tests and making decisions, versus learning the nuts and bolts of how test equipment works. National Instruments provides a complete selection of the most popular instrument drivers requested by users today. For more information or to request additional instrument drivers, visit the National Instruments instrument driver network at ni.com/idnet

Conclusion

National Instruments Measurement Studio offers fast and easy solutions for GPIB control that make your instrument control task easier. In addition to the CWGPIB control, you can use the Measurement Studio IVI ActiveX controls to control a group of similar instruments, such as scopes, and take advantage of IVI class drivers, which provide instrument interchangeability. Or if your test measurement system uses a combination of VXI, serial, or GPIB instruments, you can use the Measurement Studio VISA ActiveX control to create a program that controls all of your instruments through the same API. Refer to Application Note 169, Instrument Control from Visual Basic, at ni.com for more information about the Measurement Studio IVI ActiveX controls. Refer to Application Note 147, Serial, GPIB, and VXI Communication with Measurement Studio VISA, at ni.com for more information about the Measurement Studio VISA ActiveX control.