ICS ELECTRONICS
division of Systems West Inc.

AB48-39

IEEE 488 APPLICATION BULLETIN

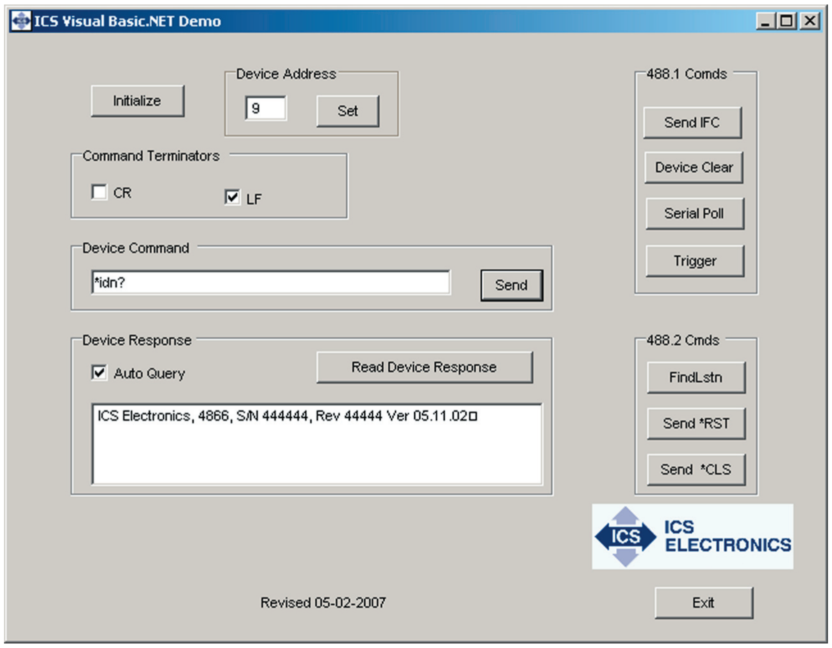
VB.NET Programming Example for GPIB Controllers

INTRODUCTION

This Application Bulletin shows how to create a Visual Basic.NET program for GPIB Controllers that utilize an industry standard GPIB-32.DLL driver library. This Demo Program was developed with Microsoft's Visual Studio 2005 and ICS Electronics GPIB-32.DLL. Earlier versions of Microsoft's .NET development systems are not compatible with ICS's included library files. While this program has only been tested with ICS's GPIB Controllers, there is no reason why it should not work on other GPIB Controllers that include a GPIB-32.DLL Driver Library.

GENERAL CONCEPTS

The Visual Basic.NET Demo Program was written as an interactive control program that the user can easily adapt to his or her use. The Demo Program has just the one form shown in the figure on the right. The form has text windows where the user can set the address of the GPIB device, enter a command string to send to the device and text window to display device responses. Other common GPIB functions like Send IFC, Device Clear, Serial Poll and Trigger are controlled by separate buttons. The Demo Program also includes the IEEE-488.2 FindLstn routine and buttons for sending the *RST and *CLS commands to the GPIB device.



Device Address: 9
Command Terminators: [ ] CR [✓ LF]
Device Command: *idn?
Device Response: [✓ Auto Query]
ICS Electronics, 4866, S/N 444444, Rev 44444 Ver 05.11.02□
Revised 05-02-2007

Visual Basic.NET Demo Main Form

REQUIREMENTS

The executable version of the Demo Program can be run on any computer that has Microsoft's Visual Studio 2005 or Microsoft's .NET Framework, version 2.0 or later, installed on it. Visual Studio 2005 is required to develop your own Visual Basic.NET program. Any new program must include ICS's GPIB-32.VB and ICSVB.VB files to link to the GPIB-32.DLL Driver Library.

RUNNING THE DEMO PROGRAM

When the Demo Program loads, only the Initialize button is enabled. This forces the user to initialize the GPIB Controller and the GPIB bus. Once the GPIB Controller is initialized, the remaining controls on the form are enabled.

Although the Device Address is preset to address 4, it can be changed manually or set automatically by running the FindLstn routine. The Demo program is designed to only control one GPIB device at a time. If you have multiple devices you will have to enter the GPIB address of the desired device into the Device Address window and press Set to select the device.

Device commands and text data can be entered into the Device command window. The Demo program will append a linefeed to the command strings and output them to the GPIB device when you press the Send button. The Demo program will automatically read the device responses if the command string was an IEEE-488.2 or SCPI query. You can manually read the device response by pressing the Read Device response button.

DEMO PROGRAM CODE

The complete Demo Program can be downloaded as a ZIP file from ICS's website. The source for the Demo Program is contained in the VB_demo.vb file which can be opened in Visual Studio or any text editor.

Visual Basic.NET is more rigorous that the older Visual Basic 6 because it requires that you preface all references to the library with a class name. However, it also provides automatic indenting and error checking as you write your program that is a big help when you are writing large functions.

The Form1 Load function only initializes a few variables and then disables all functions except Initialize.

The Initialize function outputs an IFC to clear the GPIB bus, gets the Controller's GPIB address, asserts REN and sets the timeout to 3 seconds. Ibsta is checked for an error after each GPIB call. All GPIB programs should explicitly initialize the GPIB Controller and the GPIB bus as a matter of good programming practise.

The cmdSet function is more complicated in the Demo Program than you need in a real program because of checks to verify that the user did not enter the GPIB Controller's address and to handle both primary and secondary GPIB addresses. cmdSet also checks for presence of a GPIB device at the selected address. If you use fixed GPIB addresses, you can assign a GPIB address to a descriptive variable, e.g. dvm = 04

The 488.1 commands are done by calling the appropriate function from the GPIB-32.VB library. Note that some instruments that use SCPI commands need to be enabled before they will respond to a Trigger command.

The cmdFindLstn function is the most complicated function and is presented in the Demo Program because it is very useful as a way to verify that all of the GPIB devices in your test system are present. The Demo Program creates a list of all possible GPIB addresses to test before calling FindLstn. Once you have the Device List, you can use an *IDN? query to create a table of GPIB devices vs their addresses. You can also verify that all of the GPIB devices are attached and are set to their correct GPIB addresses. Very helpful for eliminating problems when equipment has been taken away for calibration or returned with the wrong GPIB address.

The cmdSend function sets the desired termination condition and then calls the 488.2 Send function to output the command string. If the command string was a query (contains a ?) and if AutoQuery is selected, cmdRead is called. cmdRead inputs the device response string and places it in the txtResults text box.

SUMMARY

This application note has described a Visual Basic.NET program that demonstrates how to use the most common GPIB commands These commands are adequate to write most applications. The user should be able to use the Demo Program as a framework to build a complete test application.

HINT

When writing a test program, it is often handy to create a subroutine that combines cmdSend and cmdReceive as a single subroutine to reduce your coding effort. The subroutine should output the command string, check for a question mark, and if one is present read the device response. It can do any necessary string cleanup and return the cleaned response string to the calling program. An example of its use is:

```
CmdStr$ = "*ESR?"
Call Sout(dvm, Cmdstr$)
If Rdg$ <> 0 then.....
```