Using COM-based Formatted I/O in Microsoft® Visual Basic 6

Application Note

SCPI-based instrument communication in Visual Basic (VB) has often been counterintuitive to programmers familiar with the VB I/O facilities. SCPI, the Standard Commands for Programmable Instruments, describes an ASCII string language for controlling and querying instruments from PC's, usually via GPIB, LAN, USB, or RS-232 hardware connections. However, an ASCII string language is not a natural fit to VB, where general purpose formatted I/O components allow programmers great flexibility in reading, writing, and presenting data in a variety of formats. VB programmers need a similar form of instrument I/O that provides similar flexibility.

In VB, formatted I/O is the process of converting diverse data to a serial (linear stream) format such as strings of ASCII characters and also retrieving the data from those strings. VB provides locale-aware formatted I/O facilities based on the VARIANT structure, a data type that can hold any other data type in VB. On the other hand, the three most common I/O library choices for instrument communication, VISA, SICL and NI-488, provide no formatted I/O or only "C"-style formatted I/O, which doesn't work well with VB. The VXI*Plug&Play* 4.3.4 specification (*now maintained by the IVI Foundation*), "VISA COM," provides formatted I/O facilities through COM interfaces that work well in Visual Basic.

A SCPI command string may be something like:

```
SENS:VOLT:RANGE 5
```

This tells the instrument to set the voltage range for measurements to 5 volts. Users who must communicate with instruments through SCPI commands and queries often have the desire to mix fixed strings such as the "SENS:VOLT:RANGE " portion of the command to variable strings, like the number "5." This could be accomplished in VISA-style formatted I/O by executing the command:

```
viPrintf(vi, "SENS:VOLT:RANGE %d", voltRange);
```

This would substitute the value of the voltRange variable into the string and send it to the output stream. An equivalent command in Visual Basic file I/O is:

```
Print #filenum, "SENS:VOLT:RANGE"; voltRange
```

Visual Basic prefers string concatenation to format strings for I/O and display, except for the confusing and limited "Format" function. Additionally, Visual Basic does not support "C"-style variable argument lists, used by scanf and printf-style functions.

The VISA COM 4.3.4 specification defines a Formatted I/O COM component, providing the "IFormattedIO488" interface, designed for SCPI and IEEE 488.2-style communication. The VXI*Plug&Play* Alliance, now the IVI Foundation, is the group of I/O vendors that developed the VISA 4.3 specification, which defined the entry points and behavior of a DLL, "visa32.dll", which exposed a "C"-based I/O API.

The VISA COM 4.3.4 specification describes an API based on the VISA system that is exposed through Microsoft's COM technology. Since VISA's formatted I/O functions rely on "C"-style variable argument lists, which are unavailable in COM, VISA COM does not provide VISA's formatted I/O. The Alliance decided that despite this limitation and the general problem of there being no one formatted I/O style for all COM compatible programming platforms, a minimum formatted I/O library was necessary, the "IFormattedIO488" interface.

Why not use Visual Basic's formatted I/O components rather than inventing VISA COM formatted I/O? There are several reasons.

1. Many Visual Basic string conversion functions use the local language settings of the PC. For example, German language PCs use a comma instead of a decimal point to delimit the integer and fractional parts of floating-point numbers. Since SCPI compliant instruments use US English settings, using local language settings is an undesirable feature.

2. Visual Basic formatted I/O often cannot tell when to stop reading instrument data. For example, SCPI arbitrary blocks are binary structures that are often used by instruments to return large, binary arrays of data. The size of the array is sent at the beginning of the structure. VB formatted I/O does not recognize SCPI arbitrary blocks, and therefore does not know to read the size first when an arbitrary block is being read from the instrument. VB is much more likely to assume that any binary value of 10 in the block's binary array is a termination character and prematurely terminate the read.

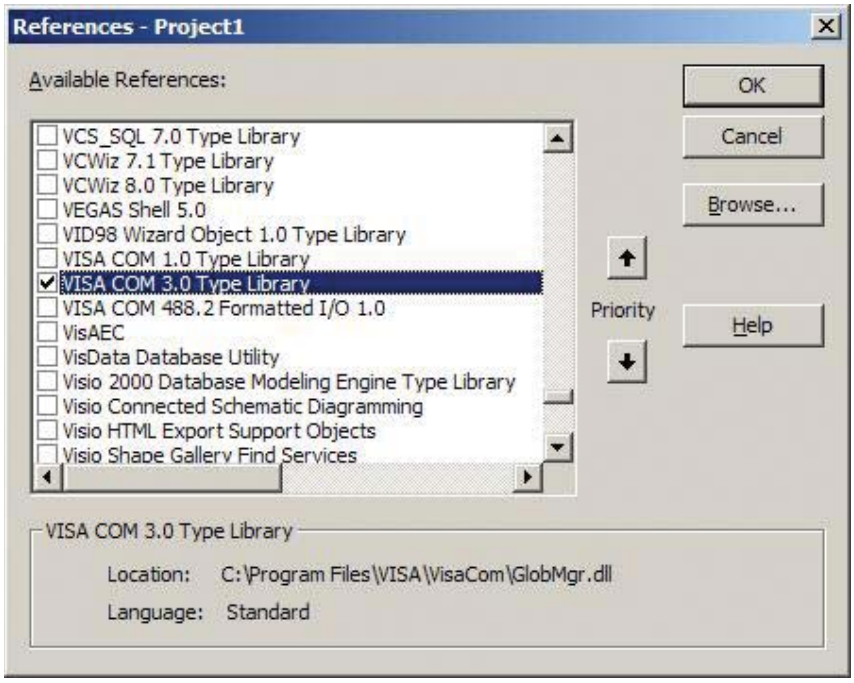
3. In general, SCPI defines complex data structures that VB cannot recognize. The binary format of the arbitrary blocks mentioned in the previous item are especially hard to use in Visual Basic, since that language does not provide easy-to-use binary and pointer manipulation operators. List formats also present some difficulties since they require tokenization.

VISA COM's IFormattedIO488 interface was designed with these problems in mind. Its design is aimed at the types of data and their formats sent and received by SCPI-based instruments as opposed to the more general design of VISA's viPrintf/viScanf facilities. It was also designed to be relatively easy to use from the most common platforms for VISA COM. Its syntax is most similar to Visual Basic's formatted I/O facilities. Since COM method call design does not allow for the complex syntax of the Print# and Input# statements, we must rely on a more function-oriented approach, where consecutive function calls build up a command to send to or retrieve data from the instrument. Ideally, this interface would be designed to provide formatted I/O that was intuitive and useful in all the programming platforms likely to be used with VISA COM. The design is a compromise that provides use patterns that work well on all the most common platforms for VISA COM: VB 6 and Visual Studio® (C++, C#, VB).

Let's try some sample I/O using VISA COM in Visual Basic 6. We'll be using the Agilent VISA COM Libraries, which can be downloaded from the Agilent website, as a part of the Agilent IO Libraries Suite, at **http://www.agilent.com/find/iolib.**

The first task is to import VISA COM into VB 6, which can be done under the "Project" menu heading with the "References" menu selection. After installing Agilent VISA COM, you can insert the proper VISA COM references so that your dialog box looks like Figure 1.

Figure 1.



You may encounter a defect in Visual Basic 6 that will first display the VISA COM 1.0 Type Library only. If this occurs, select the 1.0 Type Library and save your project. Then close and open your project again. Go back into References and you will see the 3.0 Type Library is selected.

"VISA COM 3.0 Type Library" is the reference to all the API's of VISA COM including the Agilent implementation of the API for creating Agilent VISA COM resources. After you add this reference to the project, you can look at the details of the formatted I/O syntax in VB's Object Browser.

First things first: let's make sure we have an instrument to talk to. Below is a short subroutine that opens an instrument session to an Agilent 34401A Digital Multimeter and gives it to the Basic Formatted I/O object, and then uses the Basic Formatted I/O object to write to and read from the instrument (Figure 2).

The output of this program in the US English locale is similar to Figure 3.

Three different objects were created in the above example, a resource manager, a formatted I/O object, and the I/O session itself. The resource manager is a class factory that knows how to create an object of the appropriate type given some configuration information, in this case a VISA address, "GPIB0::12::INSTR". The Open()method returns a new I/O session, which we immediately pass to the IO property of the formatted I/O object.

Once all these objects are created, we can begin to read and write data through the formatted I/O object. The WriteString method is fairly self-explanatory; we add the vbLf VB constant (the termination character in SCPI, \n) which is not necessary for VISA GPIB instrument sessions since they have the out-of-band END signal. The ReadList() and ReadString() methods take back the results from the "*IDN?" query.

The ReadList() method additionally tokenizes the result and sets each member to its appropriate datatype, so a token such as "662.3" becomes a floating-point VARIANT instead of a string VARIANT, which means if you displayed the results on a computer with a French locale setting, the number would appear as "662,3." You can customize the tokenization with the optional second parameter, a string of characters, each to be

Figure 2

```
VB 6
Private Sub BasicCommunication()
    Dim rm As New VisaComLib.ResourceManager
    Dim fmio As New VisaComLib.FormattedIO488
    Dim list(), item

    Set fmio.IO = rm.Open("GPIB0::22::INSTR")

    fmio.WriteString ("*IDN?" & vbLf)
    TextBx.Text = "The full *IDN? string=""" & _
        fmio.ReadString() & """" & vbCrLf

    fmio.WriteString ("*IDN?" & vbLf)
    list = fmio.ReadList()
    For Each item In list
        TextBx.Text = TextBx.Text & "Element Type=""" _
            & TypeName(item) & """ Element Value=""" _
            & item & """" & vbCrLf
    Next item

    fmio.IO.Close
End Sub
```

Figure 3

```
The full *IDN? string="HEWLETT-PACKARD,34401A,0,7-4-2"
Element Type="String" Element Value="HEWLETT-PACKARD"
Element Type="String" Element Value="34401A"
Element Type="Long" Element Value="0"
Element Type="String" Element Value="7-4-2"
```

treated as a list separator. The default value of this parameter is the comma and the semicolon characters.

The next example (Figure 4) is of a basic measurement:

Sample output of this subroutine in the US English locale is:

```
0.00000135
0.00000555
```

Sample output of this subroutine in the French locale is:

```
0,00000113
0.00000511
```

Notice that the second French measurement returned a US English-style number. This is because the second read method was ReadString(), which read the data into the VARIANT as a raw string, so VB's locale-aware conversion methods were never called. It is obviously preferable to use the ReadNumber() method to do the data conversion so that the computer's locale does not adversely affect program behavior.

This subroutine uses the optional second parameter of the WriteX() methods to build up a command before sending it. This parameter is called "FlushAndEnd" and it tells the Basic Formatted I/O object whether to send the data in the buffer to the instrument and send the END signal, telling the instrument the computer is done talking. By setting this parameter to false, you can build up a command before transmitting it on the last WriteX() method. The second version builds up an array of data and sends all of the data at once with the WriteList method, using the space character as the list separator. By using the formatted I/O object's methods, we avoid causing VB to do an implicit type conversion, which is what would happen if we did the following command:

Figure 4

VB 6

```
Private Sub BasicMeasurement()
    Dim rm As New VisaComLib.ResourceManager
    Dim fmio As New VisaComLib.FormattedIO488
    Dim range As Integer
    Dim resolution As Double, reading
    Dim query()

    Set fmio.IO = rm.Open("GPIB0::22::INSTR")

    fmio.IO.Clear
    fmio.WriteString ("*RST" & vbCrLf)
    fmio.WriteString ("*CLS" & vbCrLf)

    range = 1
    resolution = 0.001
    fmio.WriteString "MEASURE:CURRENT:AC? ", False
    fmio.WriteNumber range, flushandend:=False
    fmio.WriteString "A, ", False
    fmio.WriteNumber resolution, flushandend:=False
    fmio.WriteString "MA", True

    reading = fmio.ReadNumber()
    TextBx.Text = reading & vbCrLf

    query = Array("MEASURE:CURRENT:AC? ", range, _
        "A, ", resolution, "MA" & vbLf)
    fmio.WriteList query, ASCIIType_Any, " "

    reading = fmio.ReadString()
    TextBx.Text = TextBx.Text & reading

    fmio.IO.Close
End Sub
```

```
fmio.WriteString "MEASURE:CURRENT:AC? " & range & _
"A, " & resolution & "MA" & vbLf
```

The consequence of this call is that the data parameters are converted to strings via the locale-aware VB code. This would result in a bad command being sent if the computer's locale were set to a language that used a different decimal place character.

The next example uses an Agilent 54501A Oscilloscope to demonstrate reading binary data with the Basic Formatted I/O object (Figure 5).

The output of this subroutine is:

```
The data type="Byte()"
The array length=500
```

This subroutine asks the oscilloscope to return the current waveform as an array of 500 bytes in the IEEE 488.2 Definite-Length Binary block format. The command ReadIEEEBlock(BinaryType_UI1) causes the Basic Formatted I/O object to parse the block and return an array of bytes. This method will even work on RS-232 and TCP/IP socket interfaces with definite-length binary blocks (you need to disable the Termination Character before the ReadIEEEBlock call and re-enable the Termination Character after). There is an equivalent WriteIEEEBlock() that takes an array as a parameter and converts the data into a definite block to send to the instrument.

Keep in mind the endianness (byte order) of the instrument with which you are communicating when using the binary methods. It does not matter for arrays of bytes, but for integers, reals, and other multibyte data types you must know the endianness of the instrument and set the "InstrumentBigEndian" boolean property of the Basic Formatted I/O object accordingly.

These examples give a brief tutorial of the use of the Basic Formatted I/O component provided with Agilent's VISA COM. These examples also demonstrate the benefits of using the formatted I/O facilities provided with VISA COM over using the native facilities of Microsoft Visual Basic 6.

Note: These examples were created using Microsoft Visual Basic 6 and Agilent IO Libraries Suite 14.2.

Figure 5

VB 6

```
Private Sub BinaryMeasurement()
    Dim rm As New VisaComLib.ResourceManager
    Dim fmio As New VisaComLib.FormattedIO488
    Dim range As Integer
    Dim resolution, reading As Double
    Dim query, waitLoop As Long

    Set fmio.IO = rm.Open("GPIB0::12::INSTR")

    fmio.IO.Clear
    fmio.WriteString "*RST"
    fmio.WriteString ":AUTOSCALE"

    ' Use a DoEvents Loop to wait a few seconds for the
    ' scope to catch up
    For waitLoop = 1 To 600000
        DoEvents
    Next waitLoop

    fmio.WriteString ":WAV:DATA?"
    query = fmio.ReadIEEEBlock(BinaryType_UI1)

    TextBx.Text = "The data type=""" & _
        TypeName(query) & """" _ & vbCrLf
    TextBx.Text = TextBx.Text & "The array length=" & _
       UBound(query) - LBound(query) + 1

    fmio.IO.Close
End Sub
```

Agilent Email Updates

www.agilent.com/find/emailupdates
Get the latest information on the products and applications you select.

Agilent Direct

www.agilent.com/find/agilentdirect
Quickly choose and use your test equipment solutions with confidence.

Agilent Open

www.agilent.com/find/open
Agilent Open simplifies the process of connecting and programming test systems to help engineers design, validate and manufacture electronic products. Agilent offers open connectivity for a broad range of system-ready instruments, open industry software, PC-standard I/O and global support, which are combined to more easily integrate test system development.

LXI

www.lxistandard.org
LXI is the LAN-based successor to GPIB, providing faster, more efficient connectivity. Agilent is a founding member of the LXI consortium.

Remove all doubt

Our repair and calibration services will get your equipment back to you, performing like new, when promised. You will get full value out of your Agilent equipment throughout its lifetime. Your equipment will be serviced by Agilent-trained technicians using the latest factory calibration procedures, automated repair diagnostics and genuine parts. You will always have the utmost confidence in your measurements.

Agilent offers a wide range of additional expert test and measurement services for your equipment, including initial start-up assistance onsite education and training, as well as design, system integration, and project management.

For more information on repair and calibration services, go to

www.agilent.com/find/removealldoubt

www.agilent.com

For more information on Agilent Technologies' products, applications or services, please contact your local Agilent office. The complete list is available at:
www.agilent.com/find/contactus

Phone or Fax

Americas	
Canada	877 894 4414
Latin America	305 269 7500
United States	800 829 4444

Asia Pacific	
Australia	1 800 629 485
China	800 810 0189
Hong Kong	800 938 693
India	1 800 112 929
Japan	81 426 56 7832
Korea	080 769 0800
Malaysia	1 800 888 848
Singapore	1 800 375 8100
Taiwan	0800 047 866
Thailand	1 800 226 008

Europe	
Austria	0820 87 44 11
Belgium	32 (0) 2 404 93 40
Denmark	45 70 13 15 15
Finland	358 (0) 10 855 2100
France	0825 010 700
Germany	01805 24 6333* *0.14€/minute
Ireland	1890 924 204
Italy	39 02 92 60 8484
Netherlands	31 (0) 20 547 2111
Spain	34 (91) 631 3300
Sweden	0200-88 22 55
Switzerland (French)	44 (21) 8113811 (Opt 2)
Switzerland (German)	0800 80 53 53 (Opt 1)
United Kingdom	44 (0) 7004 666666

Other European Countries:
www.agilent.com/find/contactus

Revised: March 23, 2007

Printed in USA, April 18, 2007
5989-6583EN

Agilent Technologies