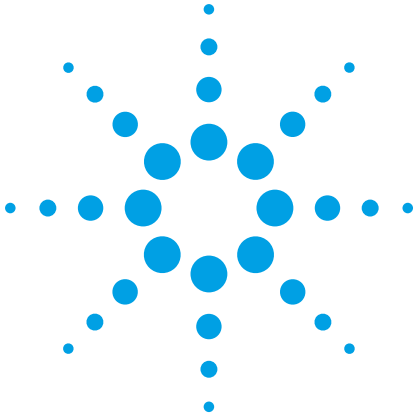
Using VISA COM I/O API in .NET



.NET and Test & Measurement I/O

The Microsoft® .NET architecture, with its C# and Visual Basic programming languages and the Microsoft Visual Studio® development environment, has many features that make it an excellent environment for Test & Measurement programmers. There is only one problem: doing actual instrument communication. Agilent has released the T&M Toolkit, which provides services, applications, and API's to make instrument programming natural and simple in Visual Studio environments (**www.agilent.com/find/toolkit**). While the T&M Toolkit provides a simple, easy-to-use programming environment from Visual Studio, savvy Visual Studio users today can fall back on more basic I/O tools such as VISA COM I/O to get their T&M programming tasks done in .NET environments.

VISA COM I/O and .NET

The VISA COM I/O API is a programming interface standardized by the IVI Foundation for communicating with instruments over GPIB, LAN, USB, RS-232, or other hardware interfaces. Agilent Technologies has an implementation of the VISA COM I/O standard that works with Agilent I/O hardware as well as the computer-standard I/O interfaces. VISA COM I/O is an update of the older VISA C API to work in and with Microsoft's COM technology.

Microsoft has integrated robust support for COM components in the .NET environment. The COM interfaces tend to translate well into .NET equivalents via automated wrapper-generator tools that Microsoft provides. Due to this COM support in .NET, many programmers will find VISA COM to be an excellent choice for instrument communication in .NET. This article describes how to use Agilent's VISA COM I/O implementation with C# and Visual Basic (VB) examples.

Getting started

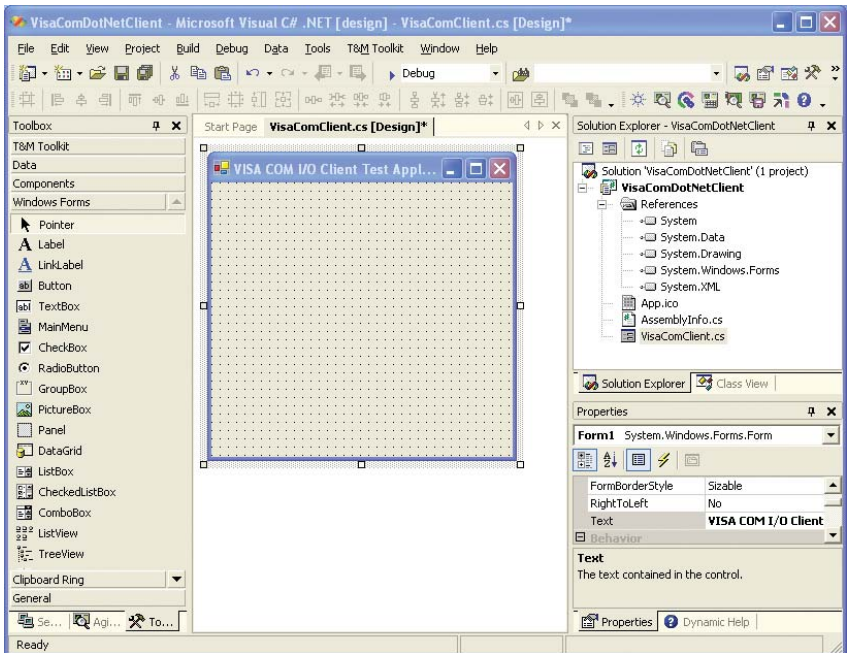
The examples in this application note were developed using C# 2003 and Visual Basic 2003 as part of Visual Studio 2003 Professional Edition. We will commonly call them C# and VB throughout this application note. In addition, Agilent IO Libraries Suite 14.2 was used.

First you must install Agilent VISA COM I/O, which is installed as part of the Agilent IO Libraries Suite. You can download the latest version of the IO Libraries Suite at **www.agilent.com/find/iolib**. After installing the IO Libraries Suite, you are ready to add VISA COM I/O to your C# Microsoft Visual Studio project. To use VISA COM I/O, you need to create a reference to it in your project. To add a COM reference to your project, click the "Solution Explorer" window in your C# project and right-click the "References" menu item.

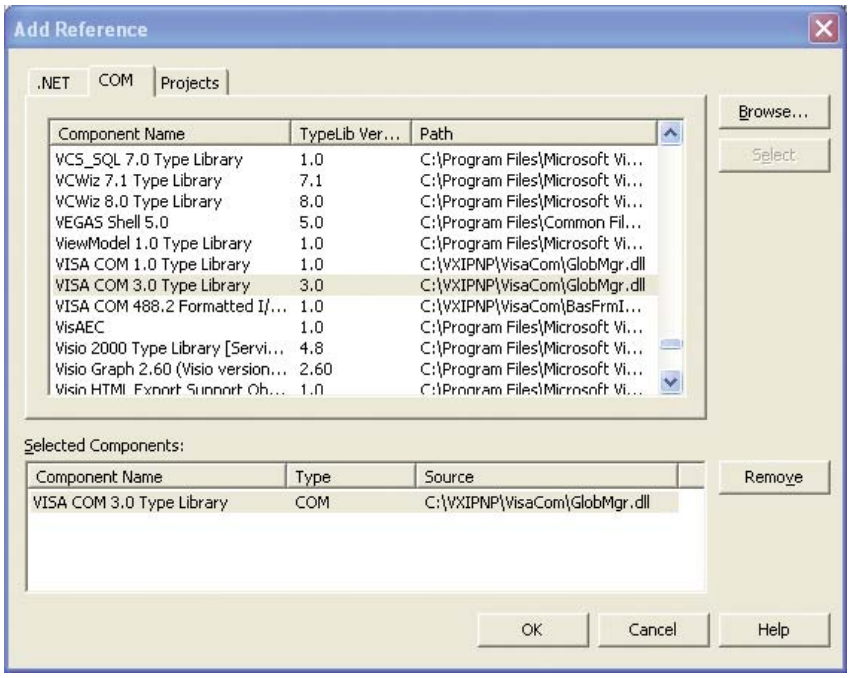
The Visual Studio Add Reference dialog

You will see the "Add Reference" dialog. Click on the "COM" tab to look for the VISA COM reference you will need. Select the "VISA COM 3.0 Type Library" reference so that you can instantiate the Agilent VISA COM I/O implementation. The VISA COM 3.0 Type Library also includes the type information for VISA COM I/O.

The Visual Studio environment with the Solution Explorer window



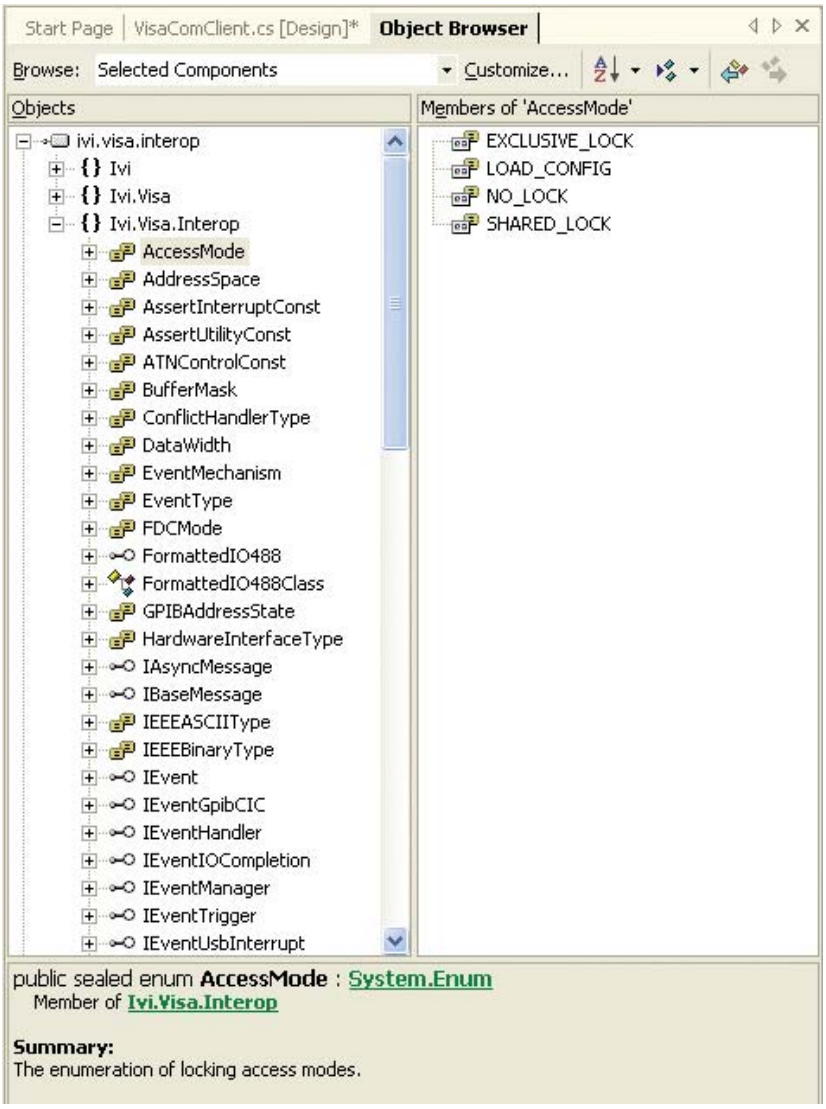
The Visual Studio Add Reference dialog



Get familiar with the Object Browser

Now that you have added the appropriate references into your application, it is a good time to look at the interfaces and classes available to use. Press `CTRL+ALT+J` or go to the View menu and select `Object Browser` to open the Object Browser. This window allows you to examine the class hierarchies of all the currently referenced COM and .NET projects and libraries. Take a look at `Ivi.Visa.Interop` to see the classes and interfaces you will be using.

The Visual Studio Oject Browser dialog



Instantiating and using Agilent's VISA COM I/O in .NET

Once you have your references to VISA COM in your project, you are ready to create and use VISA COM I/O objects. Included is an example of a simple method that creates a resource and uses the VISA COM 488.2 Formatted I/O component to communicate with an Agilent 54501A Oscilloscope. It is shown using both C# and VB.

C#

```
private void DoInstrumentIO()
{
   Ivi.Visa.Interop.ResourceManagerClass rm = new Ivi.Visa.Interop.ResourceManagerClass();
   Ivi.Visa.Interop.FormattedIO488Class ioobj = new Ivi.Visa.Interop.FormattedIO488Class();

   try
   {

      object[] idnItems;

      ioobj.IO = (Ivi.Visa.Interop.IMessage) rm.Open("GPIB2::10::INSTR",
         Ivi.Visa.Interop.AccessMode.NO_LOCK, 0, "");

      ioobj.WriteString("*IDN?", true);

      idnItems = (object[]) ioobj.ReadList(Ivi.Visa.Interop.IEEEASCIIType.ASCIIType_Any, ",");

      foreach(object idnItem in idnItems)
      {
         System.Console.Out.WriteLine("IDN Item of type " + idnItem.GetType().ToString());
         System.Console.Out.WriteLine("\tValue of item is " + idnItem.ToString());
      }

   }
   catch(Exception e)
   {
      System.Console.Out.WriteLine("An error occurred: " + e.Message);
   }
   finally
   {

      try{ ioobj.IO.Close(); }
      catch {}

      try{
      System.Runtime.InteropServices.Marshal.ReleaseComObject(ioobj);
      }
      catch {}

      try{
      System.Runtime.InteropServices.Marshal.ReleaseComObject(rm);
      }
      catch {}
   }
}
```

VB

```
Private Sub DoInstrumentIO()

   Dim rm As Ivi.Visa.Interop.ResourceManagerClass
   Dim ioobj As Ivi.Visa.Interop.FormattedIO488Class

   Dim idnItem As Object
   Dim idnItems As Object()

   Try

      rm = New Ivi.Visa.Interop.ResourceManagerClass
      ioobj = New Ivi.Visa.Interop.FormattedIO488Class

      ioobj.IO = rm.Open("GPIB2::10::INSTR")

      ioobj.WriteString("*IDN?", True)

      idnItems = ioobj.ReadList(Ivi.Visa.Interop.IEEEASCIIType.ASCIIType_Any, ",")
      For Each idnItem In idnItems

         MsgBox("IDN Item of type " + idnItem.GetType().ToString())
         MsgBox("Value of item is " + idnItem.ToString())

      Next idnItem

   Catch e As Exception

      MsgBox("An error occurred: " + e.Message)

   Finally

      Try
         ioobj.IO.Close()
      Catch ex As Exception
      End Try

      Try
         System.Runtime.InteropServices.Marshal.ReleaseComObject(ioobj)
      Catch ex As Exception
      End Try

      Try
         System.Runtime.InteropServices.Marshal.ReleaseComObject(rm)
      Catch ex As Exception
      End Try

   End Try

End Sub
```

Using Agilent VISA COM I/O in Microsoft Visual Studio

The line `Ivi.Visa.Interop.ResourceManagerClass rm = new Ivi.Visa.Interop.ResourceManagerClass()` creates the Global Resource Manager (GRM), which can instantiate (create) any VISA COM resource installed on the system. Here you see it used to open a GPIB resource at "GPIB2::10::INSTR". The line `Ivi.Visa.Interop.FormattedIO488Class ioobj = new Ivi.Visa.Interop.FormattedIO488Class()` creates an instance of the 488.2 Formatted I/O Class, which can help with parsing and writing out the data types most instruments use. Setting the IO property of the formatted I/O object prepares the object for reading and writing.

You may notice a few differences between C# and VB. These differences in large part mirror the differences between Microsoft Visual C++ 6 and Microsoft Visual Basic 6. Aside from the obvious syntactic differences there is a capability difference in how you can use VISA COM I/O. The optional parameters on the `Open()` method in VB are not optional in C#, and optional parameters in general are lost in C#.

After creating the VISA COM I/O objects to be used, you see a call to `WriteString()`. This call sends the "*IDN?" string to the instrument. The next line uses the `ReadList()` method to parse the *IDN?" return value. The method returns an object, which you can cast to an array based on the `type` parameter of the `ReadList()` Method. With type `ASCIIType_Any`, the return value is an array of objects.

The code in the Finally block is designed to clean up the I/O to be sure that the I/O session is closed immediately, all hardware I/O resources are released, and any valid COM objects are released. In COM environments like Visual C++, it was possible to destroy objects by removing the last reference to them, but in the .NET environment, you must explicitly close the session. Call the `Close()` method on the Agilent VISA COM Formatted I/O session to cause the session to release any hardware I/O resources.

Advanced VISA COM I/O operations in .NET

One of the design goals of Microsoft's COM technology was to try to simplify threading for typical COM programmers. They used the concept of Apartments, where certain threading behaviors were guaranteed so as to limit the possible multithreading interaction the programmer would have to defend against. Perversely, this made thread programming significantly more difficult in some cases. Microsoft's .NET architecture has placed the multithreading burden back on the programmer, and there are some interactions that must be guarded against when dealing with possible multithreaded situations during VISA COM I/O programming.

The programmer must worry about VISA COM I/O and threading when VISA events are used to communicate with a device. This can take the form of asynchronous I/O, handling service requests, and other VISA events. The following code demonstrates handing an SRQ event on an Agilent 34401A multimeter. Only the VB version is included for brevity.

VB

```
Public Class Form1
    Inherits System.Windows.Forms.Form
    Implements Ivi.Visa.Interop.IEventHandler

    '. . . (extra VB stuff omitted)

    Dim rm As Ivi.Visa.Interop.ResourceManagerClass
    Dim msg As Ivi.Visa.Interop.IMessage

    Delegate Sub t_srqEvent(ByVal man As Ivi.Visa.Interop.IEventManager, _
        ByVal evnt As Ivi.Visa.Interop.IEvent)

    Public Sub SrqEvent(ByVal man As Ivi.Visa.Interop.IEventManager, _
        ByVal evnt As Ivi.Visa.Interop.IEvent)

        Try
            man.Close()
            evnt.Close()
            System.Runtime.InteropServices.Marshal.ReleaseComObject(rm)
        Catch
        End Try

        MsgBox("SRQ Occurred!", MsgBoxStyle.OKOnly, "SRQ Event")

    End Sub

    Private Sub DoAdvancedIO()

        rm = New Ivi.Visa.Interop.ResourceManagerClass
        msg = rm.Open("GPIB1::22::INSTR")

        DoGenerateSRQ(msg)

    End Sub

    Public Sub DoGenerateSRQ(ByVal msg As Ivi.Visa.Interop.IMessage)

        Dim eventman As Ivi.Visa.Interop.IEventManager

        eventman = msg

        ' Reset dmm and clear DMM status registers
        msg.WriteString("*RST;*CLS" & vbLf)

        System.Threading.Thread.Sleep(500)

        eventman.InstallHandler(Ivi.Visa.Interop.EventType.EVENT_SERVICE_REQ, Me, 1000)
        eventman.EnableEvent(Ivi.Visa.Interop.EventType.EVENT_SERVICE_REQ, _
            Ivi.Visa.Interop.EventMechanism.EVENT_HNDLR)
```

```
        ' Enable 'operation complete bit' to set 'standard event' bit in status byte
        msg.WriteString("*ESE 1" & vbLf)

        System.Threading.Thread.Sleep(500)

        ' Enable 'standard event' bit in status byte to pull the IEEE-488 SRQ line
        msg.WriteString("*SRE 32" & vbLf)

        System.Threading.Thread.Sleep(500)

        ' Assure synchronization
        msg.WriteString("*OPC?" & vbLf)

        System.Threading.Thread.Sleep(500)

        ' recieve *OPC? result
        msg.ReadString(1000)

        ' set dmm to 10 volt dc range
        msg.WriteString("Configure:Voltage:dc 10" & vbLf)
        ' set integration time to 10 Power line cycles (PLC)"
        msg.WriteString("Voltage:DC:NPLC 10" & vbLf)

        System.Threading.Thread.Sleep(500)

        ' set dmm to accept 1 trigger
        msg.WriteString("Trigger:count 1" & vbLf)

        System.Threading.Thread.Sleep(500)

        ' Place dmm in 'wait-for-trigger' state
        msg.WriteString("Init" & vbLf)

        System.Threading.Thread.Sleep(500)

        ' Set 'operation complete'  bit in standard event registers when measurement is complete
        msg.WriteString("*OPC" & vbLf)

    End Sub

    Public Sub HandleEvent(ByVal man As Ivi.Visa.Interop.IEventManager, _
        ByVal evnt As Ivi.Visa.Interop.IEvent, ByVal unused As Integer) Implements _
        Ivi.Visa.Interop.IEventHandler.HandleEvent

        Dim args() As Object
        ' the threadsafe Invoke methods are the only safe thing to do in a COM callback
        args = New Object(1) {man, evnt}
        Me.BeginInvoke(New t_srqEvent(AddressOf Me.SrqEvent), args)

    End Sub

End Class
```

Advanced I/O in Microsoft Visual Studio

The `DoAdvancedIO()` method creates the I/O session and passes it to the `DoGenerateSRQ()` method, which generates a service request and enables event handling. The VB class `Form1` implements the VISA COM I/O interface IEventHandler. This class has one method, `HandleEvent()`, which is called by VISA COM I/O whenever an event the client is interested in occurs. The `InstallHandler()` method call informs VISA COM to call the form's `HandleEvent()` method implementation whenever an SRQ event occurs. The `EnableEvent()` call turns on asynchronous handler invocation for SRQ events.

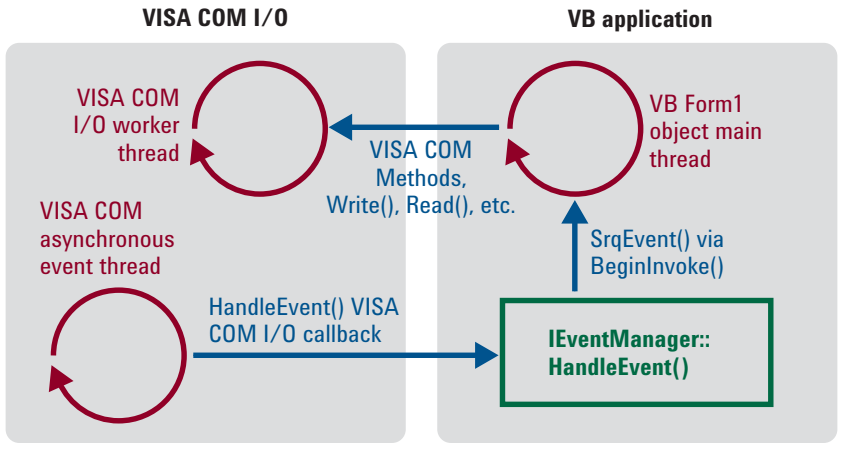
The `HandleEvent()` method does not have much code in it. This is because when you receive an event from VISA COM I/O, we don't know what thread it came in on. In VB 6, you could be certain that it was the application's main thread because VB 6 used apartment threading, guaranteeing a single-threaded environment for COM and VB methods. In an attempt to reduce overhead and complex implementation errors, Microsoft has abandoned this strategy and it is up to the user to recognize that this method call occurs in an unknown thread context. As a consequence, you must know which .NET methods are thread-specific and which ones are not, and be aware of your current threading context. Most form methods are thread-specific, so you must be careful what methods you use.

This diagram shows the thread interaction going on in the sample code above (assuming the application is multi-threaded.)

VISA COM I/O and VB threading

One of the few safe method calls in such a context is the `BeginInvoke()` method of the .NET `System.Windows.Forms.Control` class. This method accepts a .NET Delegate class, which is a wrapper for a method. We wrap up a method on the form object called `SrqEvent()` that we want to do useful things on the VB.NET Form class when an event occurs. The `BeginInvoke()` method then queues up a request on the application's main thread to call the delegate's underlying method on the form's main thread when it is free. As a consequence, you have to make sure that the application's main thread is active and not occupied in a blocking task. Also, to receive the VISA COM event at all your application cannot be blocking if it is single-threaded. In a console application, this usually means calling `Application.DoEvents()` occasionally to give queued asynchronous COM events an opportunity to execute on the application's only thread.

VISA COM I/O and VB threading



Conclusion

VISA COM I/O is a viable method of programming instruments in Microsoft's VB and C# languages, which live in the .NET execution environment. Both tools can quickly import the VISA COM I/O types for use in a relatively straightforward fashion. While the Agilent T&M Toolkit provides the best solution for simple .NET T&M programming tasks, basic and even advanced I/O operations are possible with VISA COM I/O and the Microsoft .NET programming languages.

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 LXI

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.

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

Agilent T&M Software and Connectivity
Agilent's Test and Measurement software and connectivity products and solutions allow you to take time out of connecting your instrument to your computer with tools based on PC standards, so you can focus on your tasks, not on your connections. Visit:
www.agilent.com/find/connectivity

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

United States:
(tel) 800 829 4444
(fax) 800 829 4433

Canada:
(tel) 877 894 4414
(fax) 800 746 4866

China:
(tel) 800 810 0189
(fax) 800 820 2816

Europe:
(tel) 31 20 547 2111

Japan:
(tel) (81) 426 56 7832
(fax) (81) 426 56 7840

Korea:
(tel) (080) 769 0800
(fax) (080) 769 0900

Latin America:
(tel) (305) 269 7500

Taiwan:
(tel) 0800 047 866
(fax) 0800 286 331

Other Asia Pacific Countries:
(tel) (65) 6375 8100
(fax) (65) 6755 0042
Email: tm_ap@agilent.com
Revised: 02/05/07

Product specifications and descriptions in this document subject to change without notice.

Printed in USA, March 16, 2007
5989-6338EN

Agilent Technologies