Programming USB Instruments

Application Note

Most users of test and measurement (T&M) equipment are familiar with programming an instrument over GPIB/IEEE-488. However, now that USB instruments from companies like Agilent Technologies are beginning to appear with USB interfaces, many users may want to try USB instead of GPIB. This is because of the ease-of-use and performance advantages of USB when compared with GPIB and other I/O interfaces used in T&M. This article will provide information on how to set up a test system that includes USB instruments and how to program the USB instruments using Virtual Instrument Software Architecture (VISA) I/O library software. Information is also presented about the T&M USB protocol specifications, and how these protocol specifications make use of USB endpoints. Two programming examples are available. One example shows how to use VISA to download an arbitrary waveform to the Agilent 33220A. Another example shows how to use Visual Basic to program the Agilent 33220A to execute a frequency sweep. Both of these Programming USB Instruments examples are available from www.agilent.com/find/33220a in the Drivers and Software section of Technical Support.

USB has truly become a computer standard I/O. Every new PC is shipped with the hardware and software drivers necessary to support USB, and many consumer electronic USB peripherals are available today—mice, printers, scanners, disk drives, cameras, etc. Users of T&M equipment have seen how easy it is to use these types of USB consumer electronic peripherals and have been asking T&M manufacturers to provide USB I/O on their products. With the high-speed 480 Megabits/second defined in the USB 2.0 specification, it became clear that USB would provide both ease-of-use and real performance benefits to T&M equipment users.

To illustrate the ease-of-use of USB, Figure 1 shows a simple desktop test system, similar to what might be set up on an R&D engineer's workbench. To create this test system, a USB cable simply had to be connected from the PC to the instrument.

Figure 1: Connecting a USB instrument to a PC is quick and easy

Agilent Technologies

Figure 2 shows a slightly more complex test system in which a USB hub is used. A USB hub must be used if there are no more available USB ports on the PC. A hub typically adds 4-7 additional ports and is used to create the USB "tiered" topology. USB hubs are low-cost and are readily available at any consumer electronics store. Figure 2 shows two USB instruments connected to a USB hub. Hubs introduce very little delay in USB transactions.

To illustrate the performance advantage of USB, Figure 3 shows the performance of an instrument using USB high-speed, USB full-speed, and GPIB. As can be seen from the graph, high-speed USB transfers data at 20 Mbytes/second, up to 40 times the performance of GPIB. Even full-speed USB provides about twice the performance of GPIB. The performance curves for writing to an instrument are similar.

A group of T&M equipment manufacturers recognized the potential for USB's use in T&M and began an effort in April 2001 to address the problem of how to use USB to communicate with T&M instruments. A protocol standard was necessary so that users could construct test systems using equipment from all T&M manufacturers. Any proprietary USB protocol solution would have allowed quicker time-to-market, but proprietary protocols would inevitably become obsolete and fail.

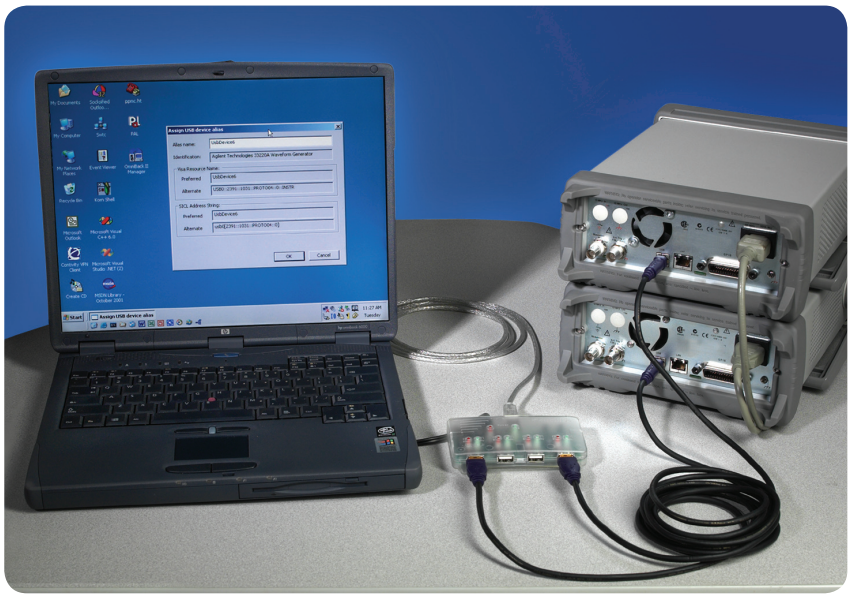

Figure 2. Using a USB hub with 2 USB instruments

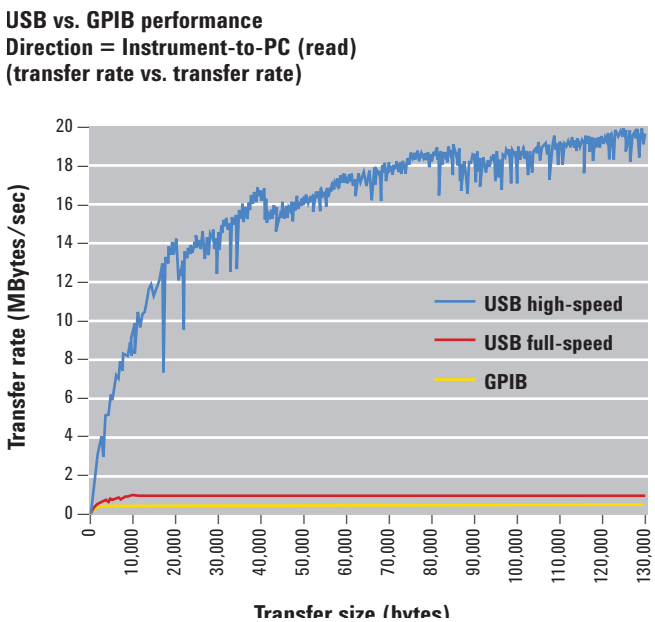

Figure 3. This graph shows that USB is up to 40 times the performance of GPIB

This group of T&M manufacturers worked within the USB Implementers Forum Device Working Group (USB-IF DWG) and started by agreeing on goals. One goal was to make it easy to modify an existing application to use USB instead of GPIB. Achieving this goal requires a mapping of GPIB "out-of-band" communications to USB. The term "out-of-band" simply refers to a communication path outside of the normal "in-band" communication path. Examples of GPIB out-of-band communications include device clear, service request, trigger, and Remote/Local. Examples of GPIB in-band communications includes program messages, which are used to set up an instrument state and to query for measurement results. In-band communication also includes the measurement results returned by instruments in response messages.

Out-of-band

Some T&M out-of-band communication, such as device clear and Remote/Local, map well to the USB control endpoint. This is because the USB specification requires a control endpoint on every device for enumeration, and it turns out that many USB Device Class specifications (Printer, Mass Storage, Still Image, etc.) also use the USB control endpoint to reset devices, which is an example of out-of-band communication.

For USB, think of an endpoint as a FIFO memory. A device can have multiple endpoints, and each endpoint has an associated address. The control endpoint address is 0. To begin a control endpoint transfer, the PC (USB Host) sends an 8-byte SETUP packet to the control endpoint. The SETUP packet contains the request and some other parameters associated with the request. The 8-byte SETUP packet may be followed by either a DATA IN phase or a DATA OUT phase. (In USB, the endpoint direction is always relative to the PC.)

Service-request out-of-band T&M communication maps well to a USB Interrupt-IN endpoint. Since USB is a master-slave protocol, the transfer of Interrupt-IN data does not really happen until the PC polls the Interrupt-IN endpoint. Fortunately, this polling interval is set by the device, but must be within the limits set by the USB 2.0 specification. For high-speed devices, the polling interval must be ≥ 1 USB microframe (125 microseconds). For full-speed, the polling interval must be ≥ 1 USB frame (1 millisecond). When polled, the device sends a Status Byte, which improves the overall efficiency of delivering interrupts and the associated Status Byte. In GPIB, after an SRQ, the PC must enter a "serial poll" sequence to find the device that pulled the SRQ line.

In-band

In-band communication of program messages sent to an instrument map well to a USB Bulk-OUT endpoint (Remember, direction is relative to the PC). Program messages are used to program an instrument state (e.g., DC or AC volts, auto-range, etc.) and to send queries to an instrument (e.g., *IDN?, *OPC?). Typical USB device silicon provide high performance bulk endpoints with larger FIFO's than the control and interrupt endpoint FIFO's. Also, USB device silicon may provide the ability to DMA from a bulk-OUT endpoint to memory, but does not provide DMA capability for control or interrupt endpoints. The ability for the PC to send a long program message quickly is important in the case of an arbitrary waveform generator. To allow devices to set up DMA, and to communicate the GPIB "End-of-message" and other "meta-data" information, each Bulk-OUT transfer is prefixed with a 12 byte Bulk-OUT Header.

In-band communication of response messages from an instrument map well to a USB Bulk-IN endpoint. Again, USB device silicon typically provides the ability to DMA from memory to a Bulk-IN endpoint. The ability for the PC to read a long response message quickly is important in the case of reading buffered A/D samples, an oscilloscope trace, or a spectrum analyzer trace. As with Bulk-OUT transfers, each Bulk-IN transfer is prefixed with a 12 byte Bulk-IN Header.

Trigger

GPIB provides both an in-band mechanism to trigger a device (“*TRG”) and an out-of-band mechanism to trigger a device (GET). Because a trigger must be synchronized with other program messages, trigger was mapped to the Bulk-OUT endpoint.

The table in Figure 4 summarizes the mapping of GPIB to USB. Examples of control endpoint, Bulk-OUT, and Bulk-IN transfers are shown later.

Figure 4. Mapping of GPIB to USB

GPIB	USB
Device clear	Control endpoint request
Remote/local	Control endpoint request
SRQ	Interrupt-IN transfer
Program message	Bulk-OUT transfer
Response message	Bulk-IN transfer
Trigger (GET)	Bulk-OUT transfer

The USB-IF DWG work concluded in December, 2002, with the acceptance of the USB Test and Measurement Class (USBTMC) base-class specification and the acceptance of the USBTMC-USB488 subclass specification. The USBTMC specification provides the ability to communicate with very simple T&M devices (sensors, A/D’s), while the USB488 specification provides the ability to communicate with more complex devices. Both of these specifications, as well as the USB 2.0 specification, can be found at www.usb.org.

VISA and USBTMC

Now that the T&M USB specifications exist, USB software can be written to match the specification. Most T&M applications make use of software that abstracts and hides all of the bus-specific protocol details for GPIB, TCP/IP, VXI, and other I/O’s so applications do not have to worry about them. This software is called Virtual Instrument Software Architecture (VISA). Agilent and other companies that make VISA implementations have made changes to the VISA specification to support the USB T&M protocol specifications. This means that an application does not have to concern itself with USB endpoints, headers, FIFO lengths, etc. The only code change to make any application run over USB involves a change to the viOpen() rsrcName parameter.

viOpen()

VISA specifies that the viOpen(...,rsrcName,...) rsrcName parameter for a USB T&M device is a string consisting of the unique attributes of the device—the idVendor, idProduct, and serial number. An example is USB0::0x0957::0x0123::SN_001001. This is rather unwieldy to type and would be prone to errors. Fortunately, the VISA specification allows an alternative rsrcName—a human readable alias. When a device is first plugged in, a VISA implementation will typically provide a dialog box that allows a user to assign this human readable alias. See Figure 5. This alias is then used instead of the idVendor, idProduct, and serial number.

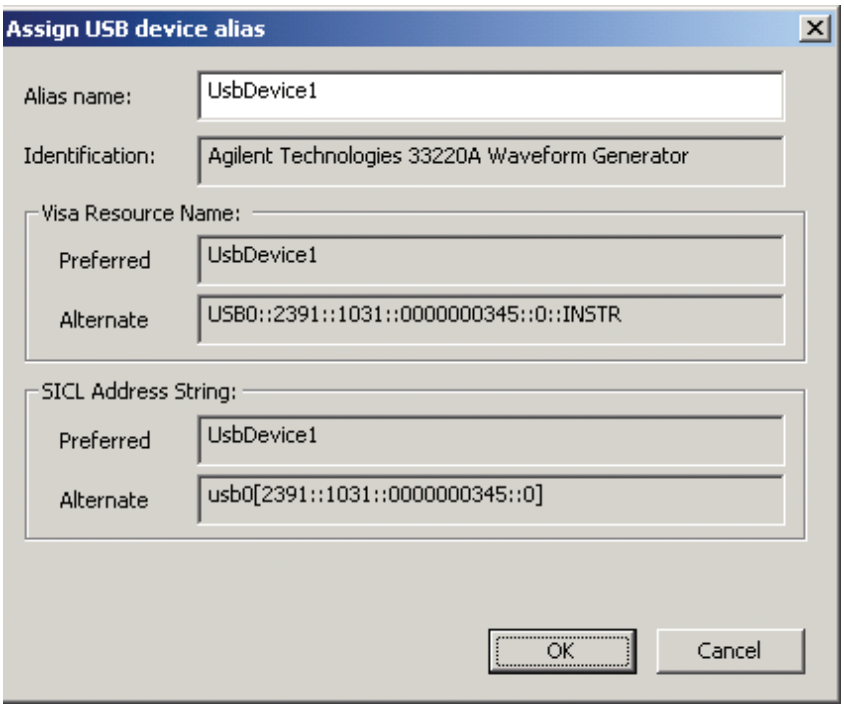

Figure 5. Assigning a USB device alias

Another significant benefit of using an alias is that the same compiled application code can run unchanged on a similar test system. This is important on the production floor, where they may be multiple identical test systems. All that must be done is to re-use the same alias names for each instrument.

viClear()

After calling viOpen(), an application might do a device clear using the viClear() API to make sure the instrument I/O is in a known state. The VISA I/O library software implements a viClear() to a USB device by sending an 8-byte control endpoint SETUP request to initiate the clear operation. This is followed by a DATA IN and finally by a 0-length DATA OUT. The complete transfer sequence to initiate a clear is shown in Figure 6.

The '05' in the 'A1 05 ... 00' sequence in the SETUP DATA0 packet identifies the request as an INITIATE_CLEAR. The '01' in the following DATA1 packet identifies the request was accepted by the device and the device has begun a clear. The 0-length DATA1 packet is required by USB and terminates the control endpoint transaction.

After the device clear is initiated, the PC must later send another control endpoint request to check the status of the clear operation. A device clear is split into an INITIATE_CLEAR and a CHECK_CLEAR_STATUS because USB requires a single control endpoint transaction to complete in 500 milliseconds, and the time for an instrument to perform a clear is potentially much longer.

viWrite()

After calling viOpen(), an application will typically send program messages to a device by calling viWrite().The VISA I/O library software implements a viWrite() to a USB device by prefixing the application buffer with a 12 byte Bulk-OUT Header and then calling a USB kernel routine to perform the write operation.

The 12-byte Bulk-OUT Header contains a message type (MsgID), a tag value (bTag), a transfer length, and an indication of whether or not the last byte in the transfer is the last byte of the message (EOM).

An example "*IDN?" query program message is shown in Figure 7.

The DATA0 '01 8B 74 ... 01 00 00 00' is the 12-byte Bulk-OUT Header and contains the meta-data for the transfer. In C-style code, this Bulk-OUT Header has the following information:

```
UINT8 MessageID;
  // identifies this as a program
  message

UINT8 bTag;
  // a tag identifying this transfer

UINT8 bTagInverse;
  // the inverse of bTag

UINT8 reserved1;
  // reserved
```

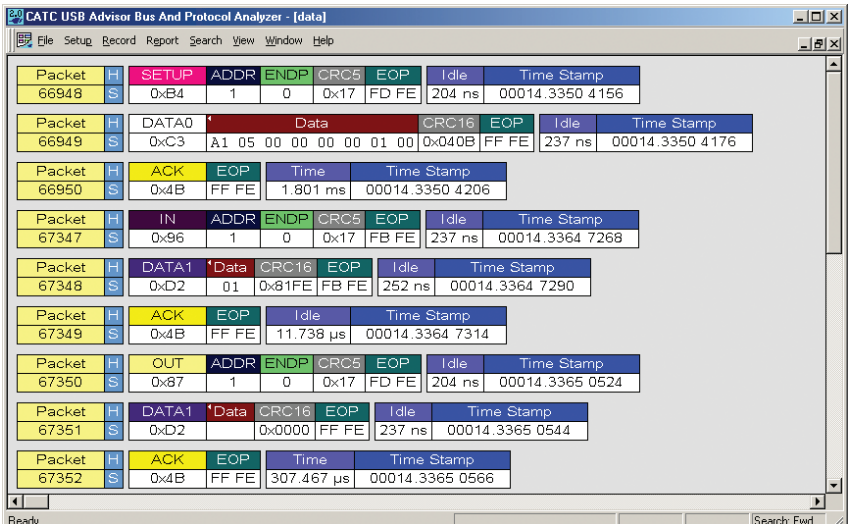

Figure 6. Initiating a device clear

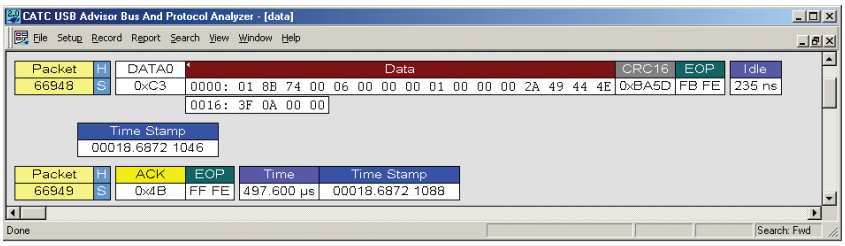

Figure 7. Sending a *IDN? program message to a USB instrument

```
UINT32 transferLength;
   // number of message data bytes
   (little-endian)

UINT8 eom;
   // bit 0=1 if last message data byte
   in payload

   // is the end of the message

UINT8 reserved2;
   // reserved

UINT8 reserved3;
   // reserved

UINT8 reserved4;
    // reserved
```

Following the Bulk-OUT Header is the hex representation of "*IDN?" followed by 2 alignment bytes. Alignment bytes to make the overall transfer a multiple of 4 bytes are required by the USBTMC specification and allow devices to DMA multiple-bytes at a time to improve performance.

viRead()

After sending program messages to set the state of an instrument, an application will typically query the instrument for a measurement result. To read the response message from a device, the application calls viRead().The VISA I/O library software implements a viRead() to a USB device by first sending a 12-byte Bulk-OUT Header that indicates how many bytes the device can send. This simplifies I/O library software since response message data bytes never have to be cached on the PC. After sending the Bulk-OUT Header, the VISA I/O library calls a USB kernel routine to perform the read operation. This will cause Bulk-IN requests to the device.

An example viRead() showing both the Bulk-OUT Header and the Bulk-IN transfer is shown in Figure 8. This example shows the identification string response to the viWrite() of the "*IDN?" query shown earlier.

The DATA1 '02 91 6E ... 00' is the Bulk-OUT Header. Note that MessageID = 0x02. This MessageID means that bytes at offset 4 to 7 represent the number of bytes (512 = 0x200, least significant byte 1st) the device may now send to the PC.

The DATA0 '02 91 6E 00 ... 01 00 00 00' is the required 12-byte Bulk-IN Header, and is followed by the '41 67 69 ... 30 0A' is the identification string, shown in hex.

viTrigger()

To cause an instrument to trigger, an application uses the VISA viTrigger() API. The VISA I/O library software implements a viTrigger() to a USB device by sending a Bulk-OUT transfer as shown in Figure 9.

The 80 in the DATA0 '80 A2 5D 00 ... 00' identifies the Bulk-OUT transfer as a trigger request. The device must execute this request in time-order with other Bulk-OUT transfers.

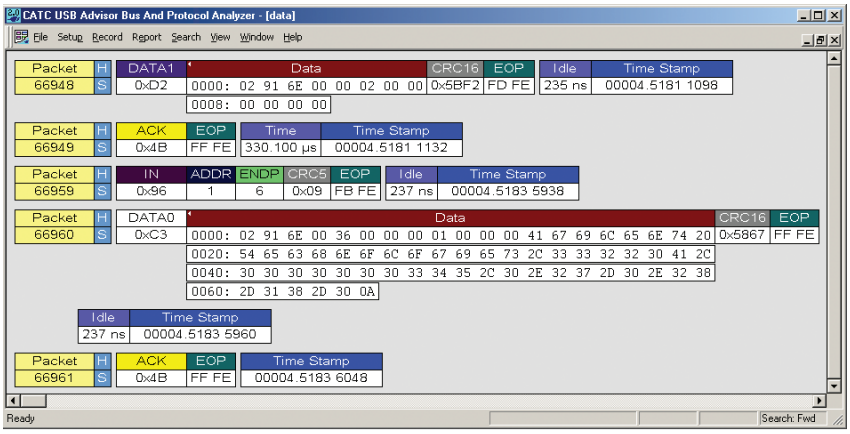

Figure 8. Reading a response message from a USB instrument

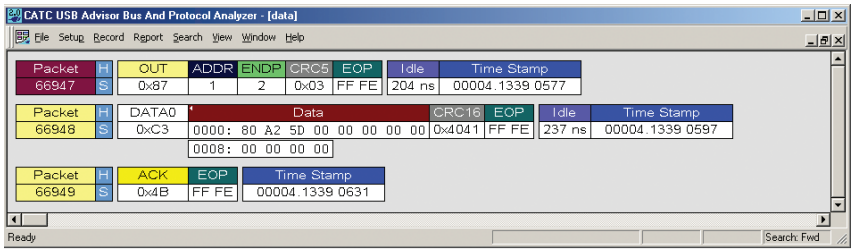

Figure 9. Triggering a USB instrument

Status Byte and VISA Events

Some applications are written to poll (busy-wait), reading the device Status Byte periodically, to detect when the device has a message available or some other status condition. Other applications may install an “event handler” using the VISA viInstallHandler() API. The specified event handler will be called by VISA when a specified event occurs.

For those applications that poll, the application can use the VISA viReadSTB() API to read a Status Byte from an instrument. The VISA I/O library software can read the Status Byte by executing a control endpoint request or use a cached Status Byte value.

For those applications that install an event handler and specify VI_EVENT_SERVICE_REQ, VISA must cause the kernel USB driver to execute IN requests to the Interrupt-IN endpoint on the device. When an SRQ condition exists, the device sends a data packet as shown in Figure 10.

The 81 in the DATA1 ‘81 50’ sequence identifies the packet as an SRQ and VISA interprets the 2nd byte as the GPIB defined Status Byte.

Conclusions

The ease-of-use and performance benefits of using USB in T&M applications will undoubtedly lead many test application developers to try USB. For those that do, they will find it easy to physically construct their test system and once in operation, they will see decreased test times.

The details given concerning the USB-IF DWG specification for T&M devices has hopefully shown how GPIB semantics have been mapped onto USB. The information about VISA has shown that using USB in a T&M application is similar to using VISA for any other interface.

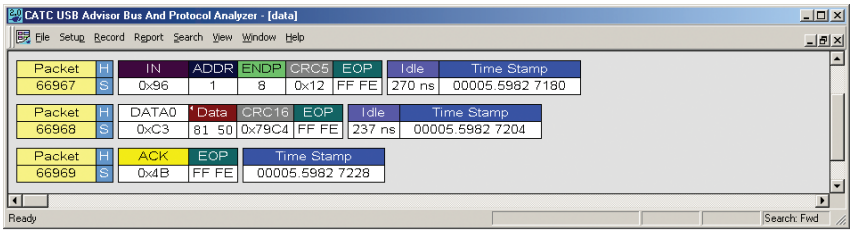

Figure 10. A USB instrument sending an SRQ

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, March 30, 2007
5989-6582EN

Agilent Technologies