Test-System Development Guide

Understanding Drivers and Direct I/O

Application Note 1465-3

This application note is part of the Test-System Development Guide series, which is designed to help you quickly design a test system that produces reliable results, meets your throughput requirements, and does so within your budget. This application note answers common questions about the use of drivers and direct I/O to send commands from a PC application to the test instrument. It discusses how the driver came about, what the different software layers do in a system to help the instrument communicate to the PC, which drivers are compatible with various software languages and I/O software, and references for further study. See the list of additional application notes in the series on page 13.

Table of contents

Agilent Technologies

Introduction

In a September 2001 survey, Test & Measurement World published a summary of engineers' worst headaches. Instrument drivers topped the list. Instrument manufacturers and various trade groups have been working on driver standards for some time, in an attempt to alleviate the frustrations of engineers who need to automate measurements and create test systems on a deadline. As a result of these efforts, we might expect finding and using appropriate drivers to be dramatically easier, but at the moment, complexities and incompatibilities are still troublesome.

This application note answers common questions about the use of drivers and direct I/O to send commands from a PC application to the test instrument. It discusses how the driver came about, what the different software layers do in a system to help the instrument communicate to the PC, which drivers are compatible with various software languages and I/O software, and references for further study.

With new insight into these topics, you should be able to choose, install and use drivers more easily and reduce the amount of time you spend getting your instruments and computer applications to talk to each other.

History

By computer standards, 1970 could be considered the mists of antiquity. That's when instruments were connected via imaginative schemes to devices resembling computers. One popular I/O format involved connecting a large cable to the instrument. Each line on the cable represented a function or range, and the line was simply grounded at the proper time. The device, say a voltmeter, would return a value using binary coded decimal (BCD) 1-2-4-8 format, or a quainter 1-2-2-4 format. Needless to say, the programming syntax of instruments at this time was anything but standardized. However, since everything was hard-wired, the process was straightforward and immediate.

GPIB

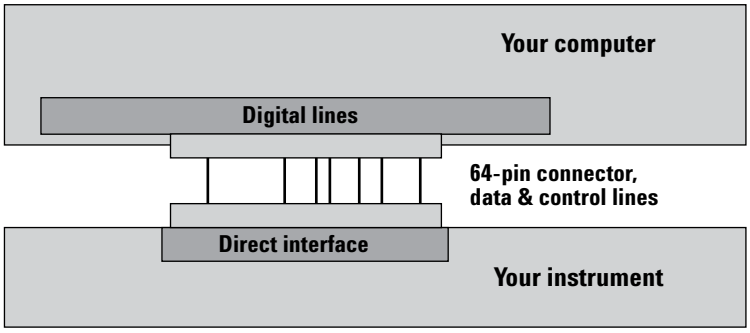
In 1971, development began on a standard hardware interface. The idea was to be able to trigger multiple instruments at once and still allow both slow and fast instruments to "talk" on the same bus without conflict. The first products to use this bus were released in 1972. The same year this new bus was dubbed Hewlett-Packard Interface Bus (HP-IB). In 1975, IEEE adopted it as a standard with little modification, and IEEE-488 was born. A variant of the original interface is now popularly known as General Purpose Interface Bus (GPIB).

With GPIB and a desktop computer (actually at the time it was called a 'desktop calculator'), the need arose for a common communication language. Limited processing power in the 'calculators' demanded a simple syntax, so ASCII commands were chosen. A DMM might be sent what was affectionately termed "R2D2 code". Here's an example:

```
"F1R2T1"
```

The command means "Go to the dc volts Function, the 1 volt Range and Trigger a reading." Different manufacturers had unique ways to interpret the command strings, based on their instruments' capabilities. If you had to replace a product with one from another manufacturer, or even a new-generation product from the same manufacturer, it could mean completely rewriting the entire program. Later versions of IEEE 488 elevated the standard from being a hardware-only standard to one that also specified protocol.

Figure 1. Early instrument control utilized hard-wired commands.



SCPI

In 1989, seeing a need for more clarity and interchangeability that was available with simple ASCII, Hewlett-Packard introduced a programming language known as Test & Measurement Systems Language (TMSL). Within less than a year, nine T&M manufacturers had met to generate a universal approach to instrument control, using TMSL as the basis. The outcome was Standard Commands for Programmable Instruments (SCPI).

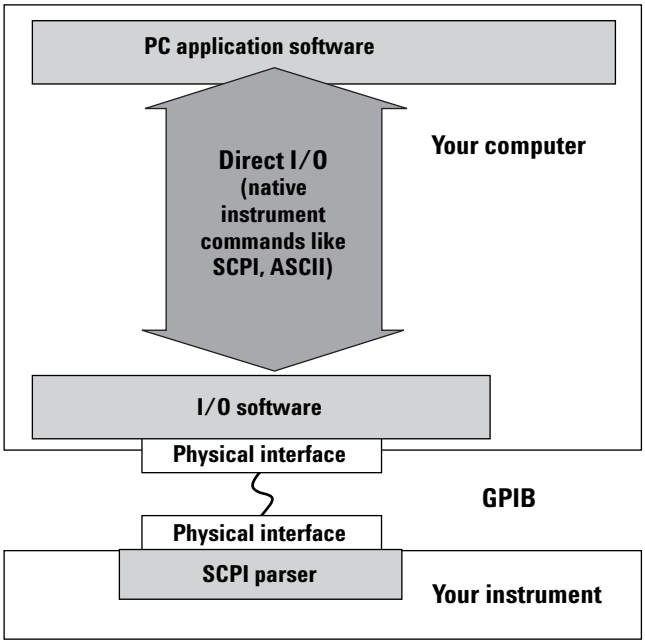
Today, SCPI is still the most-used form of instrument control. In SCPI, the instrument programming syntax became much more robust and predictable. SCPI defined a strict hierarchy, and every command was associated with a concomitant response. These were defined for source, sense and switch devices. Here's an example of SCPI code:

```
CONF:VOLT:DC 0.3,0.003
```

This command tells the instrument to configure itself to get ready to read a 0.3 volt dc signal with 3-millivolt resolution. It should be obvious from this statement that SCPI commands require some intelligence on the other end of the wire, as not every voltmeter has a 0.3 V range. The commands need to be parsed by the voltmeter and this parsing adds a small layer of delay time to the system.

One advantage of SCPI is that the list of commands typically covers 100% of the instrument's programmable functions, no matter how arcane. For a friendly tutorial on SCPI, go to: **http://ftp.agilent.com/pub/mpusup/pc/iop/hpibtut/ib5_scp.html.**

Figure 2. Compared to "R2D2" code, SCPI commands standardize programming and make life easier for the programmer. SCPI commands can access virtually any programming function in the instrument, but the parser does add small delays to the process.



The I/O Software... SICL and VISA

Instrument commands aren't the whole story. It takes more "layers" of software to communicate with a computer. Before you send the instrument a command, you need to define the I/O path, route the information through the proper I/O card, find out where the instrument is on the bus and speak to the instrument in the syntax of the I/O you're using. Assuming the GPIB I/O card in the computer is at address 7 and the DMM is at address 22 on the bus, the simple BASIC command might be:

```
ASSIGN @Dvm to 722 !
```
This tells the computer where to send the command

```
OUTPUT @Dvm;
"TRIG:SOURCE:INT" !
```
sets the trigger source to internal

The above will work with a GPIB interface, but if you try the same thing using RS-232, the syntax is very different. Switching between GPIB and RS-232 would require rewriting some code.

SICL

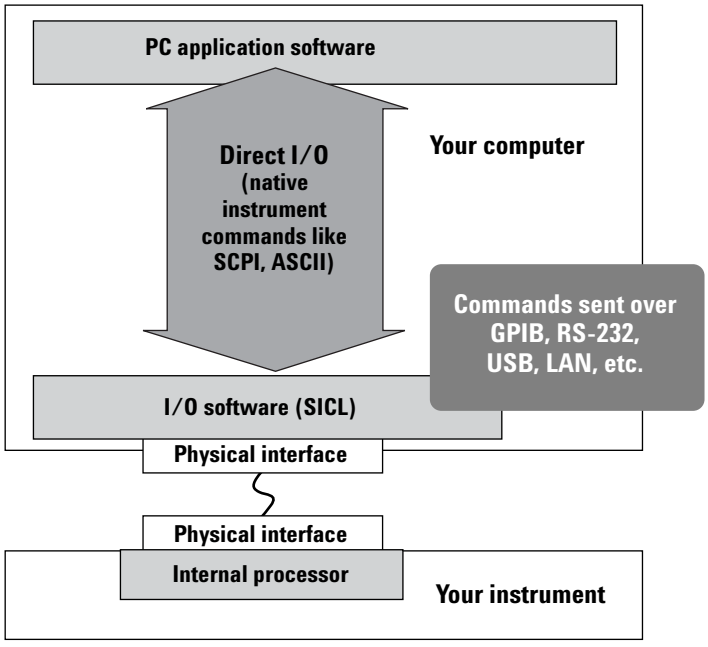
That's where Standard Instrument Control Library (SICL) I/O software comes in. SICL was developed by HP to make software as I/O-independent as possible. It adds a layer on top of the instrument code. The layer checks to see what I/O is used and alters the syntax accordingly. The code looks the same, regardless of I/O type. All you have to do is use one line of code to declare the I/O type at the beginning of the program.

SICL is not the only I/O software available today. AGILENT VISA, NI-VISA and NI-488 and VISA-COM (from Agilent) perform similar functions. That's a dizzying array of choices, so for now let's concentrate on VISA. While SICL software was created to communicate with Agilent interfaces only, VISA was created to work industry-wide.

VISA

In the late 1980's, there was a move to build standardized card cage instruments. This movement led to a software and hardware standard known as VME Extensions for Instrumentation (VXI). Based on the VME standard, VXI made special modifications for software, shielding, triggering, power supplies and analog performance. VXI was adopted by hundreds of instrument manufacturers who produced a wide variety of plug-in cards. VXI's interchangeability at the card level brought about the need for common I/O software, similar to HP's SICL, but implemented as an industry-wide standard. Largely derived from the SICL library, the VISA syntax was born.

Figure 3. SICL I/O software reduces a test engineer's programming burden by making it easier to change I/O types (USB, GPIB, USB, VXI, RS-232, etc) without recoding the program. SICL adds a software layer, which has a small effect on system speed.



Virtual Instrument Software Architecture (VISA), was created by the VXI*plug&play* Foundation to standardize I/O software across physical interfaces and between various vendors. In most cases, test systems are not solely VXI, but rather hybrids of VXI and Rack & Stack architectures, so it was not enough to create I/O software exclusively for VXI. For that reason, the VXI*plug&play* specifications were extended to include traditional standalone instruments as well as both types [1] of VXI instruments.

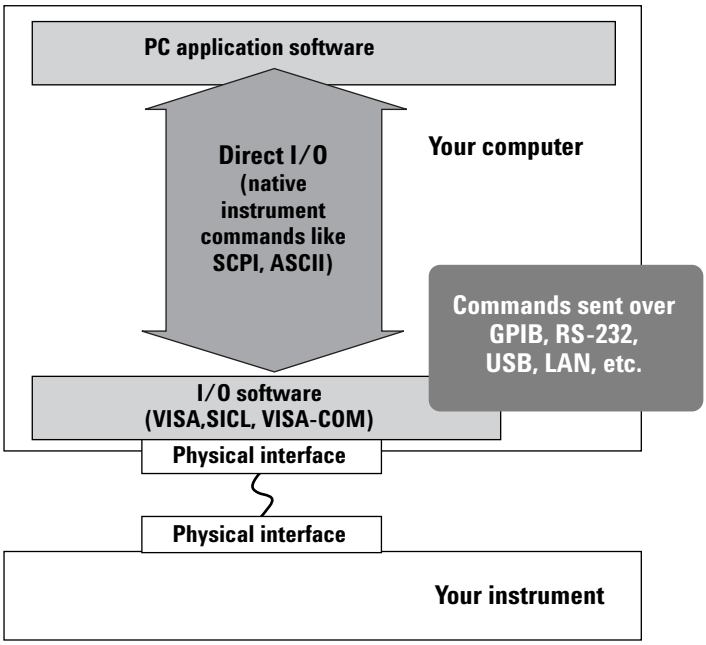
Today's two main suppliers of VISA are Agilent Technologies and National Instruments. (In 2000, the same people from HP Test & Measurement who were involved in instrument connectivity were split from HP in the new venture now known as Agilent Technologies.)

VISA I/O software uses common terminology and syntax to connect to and control instruments. A VISA library supports complete control of instrument across the physical interfaces GPIB, RS-232, USB, LAN and VXI.

The VISA library provides the capability of SICL, in a way that conforms to industry standards. A program written to work with Agilent's VISA library will work with implementations of VISA from other vendors. For those accustomed to using SICL, Agilent's implementation of VISA is provided along with its SICL libraries. (Since the introduction of VISA, programming based on the SICL library has gradually been phased out in favor of the industry-standard VISA library.)

To program a new test system, the test engineer installs the appropriate I/O library along with the application programming language. VISA was originally developed to be used with C and C++, but can also be called from any language that can call arbitrary Windows dynamic-link libraries (DLLs), including Microsoft® Visual Basic. Agilent provides header files to facilitate the use of VISA in Visual Basic .NET and C#. These can be downloaded from **http://www.agilent.com/find/iolib.**

Figure 4. VISA is the most popular form of I/O software. Drawing heavily on the work done for SICL, VISA was created to serve multiple T&M suppliers and be a universal standard. VISA-COM is a new variant of VISA.



1 VXI has two types of instruments, based mostly upon their local intelligence. "Message-based" cards" can react to a high-level message, and usually have on-card parsing. "Register-based" cards are just what the name implies... cards that have directly-programmable registers. Message-based cards do more, but are inherently slower, since they must interpret complex commands.

PC industry adds language independence

As I/O development was proceeding in the T&M industry, the PC industry was making big strides in I/O-independence and language-independence. In 1994, Microsoft stated: "The Component Object Model (COM) is a software architecture that allows components made by different software vendors to be combined into a variety of applications. COM defines a standard for component interoperability, is *not dependent on any particular programming language*, is available on *multiple platforms*, and is extensible." [2]

In February, 2001, Microsoft introduced .NET, their 3rd generation of component technology. .NET has been applied to their integrated development environment, Visual Studio® .NET, as well as MS Office, other applications, operating systems and web services.

All this is well and good, but should the Test & Measurement industry embrace PC Operating Systems?

Detractors point out the frequent operating system upgrades in the PC industry relative to T&M languages. However, from Figure 5, it can be seen that COM, which is integral to .NET components, has been around longer that most T&M standards. It seems only logical to take advantage of the investments Microsoft has made to create this paradigm shift. With 3,000 engineers working for three years on the first version of .NET, Microsoft's investment is twenty times that of the leading T&M language. Similar correlations apply to software. Visual Basic has over 6,000,000 users and C/Visual C++ has 1,000,000 users worldwide. This will result in an unprecedented body of software the average engineer will be able to leverage.

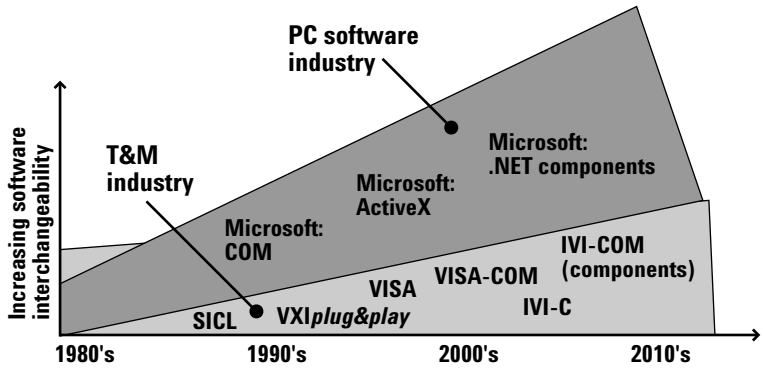
The most important immediate benefit for the test engineer is that, using Visual Studio .NET, engineers are reporting 20-30% less development time to create their test programs. They are delighted in their ability to pull in legacy code from languages such as C, Visual C++, VEE and Visual Basic into the .NET environment.

VISA-COM

To incorporate this programming language independence, Agilent initiated a VISA-COM standard as a companion to the VISA specification. VISA-COM software makes VISA services available in a language-independent COM component architecture. What does that mean? It means not only are you free to pick from popular I/O configurations, but now you also have the freedom to choose from a list of software languages like C++, C# and VB.NET. With Agilent's T&M Programmer's Toolkit product acting as a T&M "face" for .NET, you can access all this from a single environment.

When using Agilent VISA-COM, you also need to install Agilent VISA. Agilent I/O libraries are shipped along with Agilent software and I/O products.

Figure 5. PC Software Overtakes T&M Software in interchangeability. The millions of people using Visual Studio software will afford the engineer an unprecedented pool of available intellectual property.



2 Dr. Dobb's Journal, Microsoft Corp. December, 1994.

What is a driver?

It's about time we explained what a driver is; after all, that's the title of this application note. By now, we know this much: The computer has an operating system, say Windows® XP, under which there is an Application Development Environment (ADE) like Visual Studio.NET. Some language, say C#, is used to program commands for the instrument, and those commands are passed to the I/O software, which then passes them via a physical interface to the instruments' internal microprocessor. The microprocessor decodes those commands using its internal I/O structure, and the instrument carries out the commands.

To make all this practical, you need to write some code. If you are a programmer, you must either memorize or look up the Direct I/O SCPI commands related to the particular instrument being programmed. If you intend to code in a proprietary language, then you need to know how those commands fit. For simple applications, this approach works well, but as application complexity increases, using direct I/O quickly can become difficult and time consuming. Programming a direct communication path usually requires you to know a specialized computer programming language and its programming environment, and be familiar with proper command sequences and interrelationships between commands. You also need to know how to load and configure various I/O libraries and parse instrument responses that may be in the form of binary data or screen graphics. Whether you have these competencies or not, when today's product design cycles are measured in months rather than years, it doesn't make sense to spend several of those months coding a new test system, unless very high volume production is the goal.

The driver is...

The driver is a high-level, intelligent, instrument-specific or instrument class-specific piece of software intended to make programming simpler and shorten development time. In the T&M world, it facilitates communication to an instrument by by guiding the user through the steps. Its user interface can take many forms. A driver could be a list that pops up when you hit the next "dot" in Visual Basic, or it could be as elaborate as a "panel driver" that displays a virtual front panel on the screen of your computer to help you set up the instrument.

Figure 6. Agilent's T&M Programmers Toolkit using a VXI*plug&play* WIN32 power supply driver in VB .NET after being wrapped by the Driver Wizard.

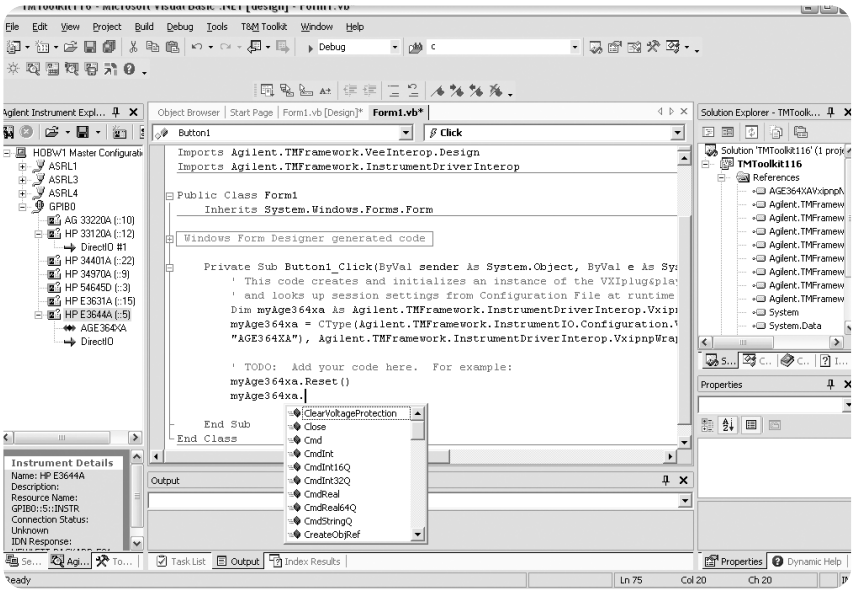
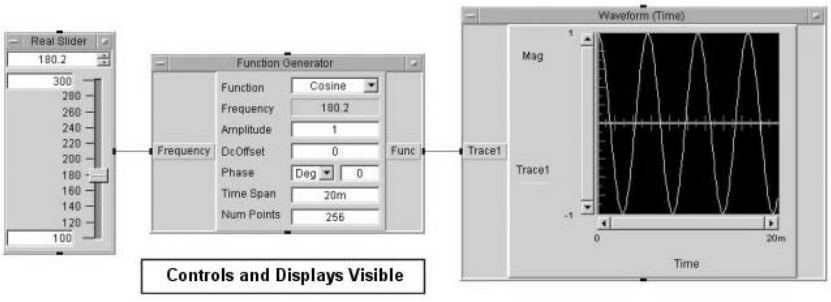


Figure 7. A tiny but interesting program, written in VEE. With its intuitive interface, VEE is the fastest T&M graphical language to learn. Fill in the boxes, and the VEE panel driver generates code for you. See **http://we.home.agilent.com/upload/cmc_upload/tmo/downloads/E206HPVEE_TESTENGR_EVAL.pdf**



Even if you have never programmed an instrument in a test system, you have probably used a driver. Digital cameras, external hard drives and printers—all require a driver to talk to the PC. If you've upgraded a PC, you may have found that the old printer driver no longer works with the new operating system, and you need to go to the Microsoft website to find a new one. Or you may find that the printer doesn't work exactly the same way it did under the old operating system. Similar issues exist in T&M equipment.

Driver coverage

A simple DMM may have only 25 commands, while a more complex instrument may have hundreds. You can imagine how expensive it is to write an intelligent driver that anticipates all the possible permutations of instrument setup, triggering, sourcing and measurement. And that's why you'll seldom see a driver that covers every command in the instrument.

Instrument manufacturers take their best guess at the commands you are likely to use and craft the driver accordingly. A typical IVI driver covers about 40-60% of the instrument's command list. This may sound like a small number, but consider this: Agilent surveyed customers who used our 3852A Data Acquisition/Switch Unit. It was a complex instrument with over 300 distinct commands available. By poring over our customers' code, we found they rarely used more than 5% of the available commands. This is an extreme case, but it tells you that 40%-60% coverage is a good start.

Figure 8. The driver is, among other things, a programming aid that works between the PC application and the I/O software. It can save enormous amounts of development time and prevent mistakes, but can also slow system performance by adding another layer of software.

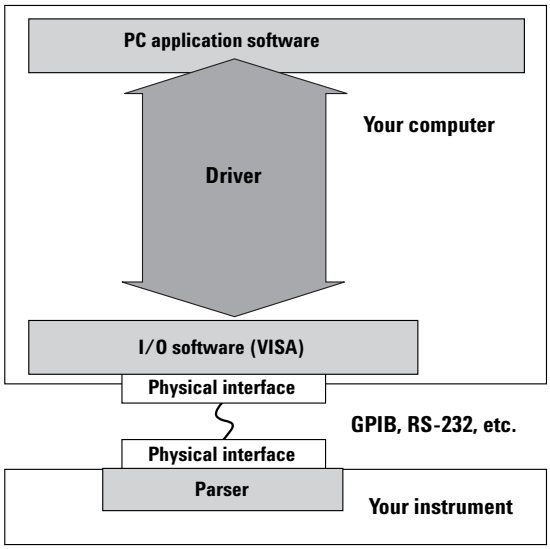
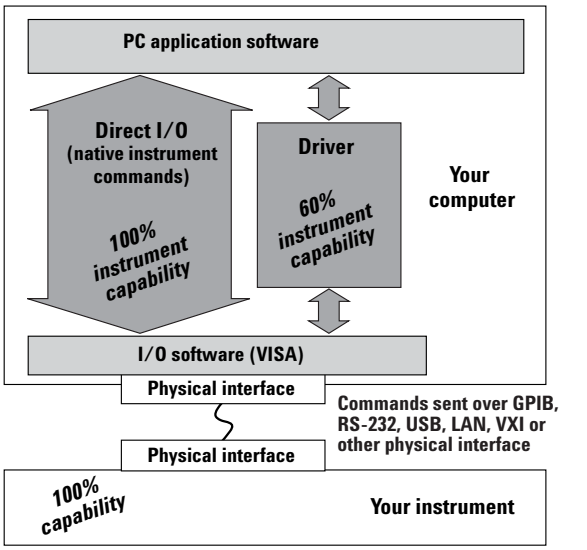


Figure 9. If you are using a driver and need to access instrument functions the driver doesn't have, you can send direct SCPI or ASCII commands, or go through the driver with pass-through commands to control the instrument directly. This gives you the convenience of drivers, with the 100% coverage of direct I/O. To avoid command conflicts, this technique requires in-depth knowledge on the part of the programmer.



Generations of drivers

There are three basic generations of drivers: **Proprietary T&M drivers, Traditional T&M drivers and Component PC drivers** (Figure 10). These represent the past, present, and future of driver technology. In the past, instrument drivers were custom-designed to function with a vendor's own application development environment (ADE). A considerable body of legacy application programs uses these proprietary drivers, but for new development, engineers today have better choices.

When you need to accelerate test system design and deployment, Agilent recommends the new IVI-COM driver and the VXI*plug&play* WIN32 driver for instrument control. The only Component PC driver built on PC standard architecture is the new IVI-COM driver. This standard is being led by Agilent and other instrument companies. A component driver built on COM works in all popular PC languages and most T&M languages, uses the most popular types of I/O, can be used in the latest .NET technologies and is backward-compatible.

What is IVI?

Notice the word "IVI" is sprinkled around the chart in Figure 10. In 1998, test and measurement companies formed the **Interchangeable Virtual Instrument (IVI) Foundation** [3] to address the high cost of developing and maintaining test system software and being able to evolve technology more rapidly, by the use of better drivers. The foundation comprises end-user test engineers, equipment manufacturers and system integrators with many years of experience building test systems.

IVI classes:

The goal of hardware interchangeability led IVI to the concept of instrument classes. The idea is as simple as it sounds: If you use a spectrum analyzer, it certainly would save time if you could program every instrument in the spectrum analyzer class the same way, no matter who built it. Both the specification and any specific driver that implements it are called an **IVI Class Driver** (IVI-C Class or IVI-COM Class).

As of this writing, the IVI Foundation has defined the following instrument classes: DC Power Supply, Digital Multimeter (DMM), Function Generator/Arbitrary Waveform Generator, Oscilloscope, Power Meter, RF Signal Generator, Spectrum Analyzer and Switch. Others are under development.

This work makes it much simpler for the engineer to program instruments from separate suppliers, when those instruments conform to a particular "class".

Figure 10. The three generations of drivers represent varying degrees of language independence. IVI-COM is the newest and the one supporting the widest variety of software environments.

Instrument driver families

Component PC (based on PC standards)	**Traditional T&M** (based on T&M standards)			**Proprietary T&M** (specific to one language)	
IVI-COM	**IVI-C (via NI)**	**WIN VXI*plug&play***	**LabWindows/ CVI Plug&Play**	**VEE Panel Drivers**	**LabVIEW Plug&Play (VXI*plug&play* GWN)**

3 For additional information, you can visit the IVI Foundation website at: ***www.ivifoundation.org****.*

When should I use a driver?

Use an instrument driver if:

- A driver is available that works with your development environment and I/O software, and supports the majority of instrument features you want to use.
- You want easy access to commonly used instrument functions because the instrument commands are typically organized in a hierarchical structure
- You want to simplify the process of developing and maintaining your code over time, because there is a single point of interface to update or change
- Software interchangeability is important to you.
- You need to simplify maintaining the system when instruments need to be exchanged.

Use direct I/O if:

- You have instrument programming experience or access to programming experts
- You need to use instrument features not supported by the available drivers (the other 40~80% of the instrument capability)
- You need the absolute maximum in system throughput speed
- You need to control the exact configuration of the instruments in your system
- You have a large volume of legacy SCPI-based code.

Conclusion

If the project you are pursuing is not complex, there are often situations where you don't even know you are using a driver. Indeed, that is the ultimate goal of T&M companies... to keep this process entirely transparent. In the meantime, if you do get embroiled with issues of driver selection, note there can be tradeoffs between speed of development and speed of execution. The industry is working through these issues by instituting faster I/O and software aids, such as tools to keep track of instrument states. The whole idea is to give you both fast programming times and fast throughput.

If you choose to use a driver, computer industry-standard IVI COM drivers and a Visual Studio .NET-compliant development program such as the Agilent T&M Programmers Toolkit give you significant leverage. The T&M applications you develop will show significant hardware and software interchangeability, while being easily maintainable and extensible. The intellectual property you create during the development process will be widely transferable to other projects.

For downloads or more information on drivers, I/O software, connectivity and application software, join us at the Agilent Developer Network: **www.agilent.com/find/adn.**

Appendix

Resources

Where do I get drivers and driver tools?
Instrument vendors typically provide drivers on a CD with new products and offer their most up-to-date instrument drivers on their Web pages. Table 1 lists some of the primary sources.

Third-party software and systems integration companies that support the test-and-measurement industry can provide driver development tools and services. One such company is Vektrex **(www.vektrex.com)**.

Agilent offers its own drivers on the Web at **www.Agilent.com/find/ADN**, but it does not post drivers written by others. Because you are at the mercy of whoever created the driver, it is a good idea to use a driver supplied by the same vendor who made the equipment.

Tools

Mixing I/O hardware and I/O software from different suppliers
Want to use Agilent I/O cards with NI LabVIEW software?

Want to use NI I/O cards with Agilent VEE?

Need to install Agilent VISA and NI-VISA side by side?

Help is available for all these scenarios. Go to: **ftp://ftp.agilent.com/pub/mpusup/pc/binfiles/iop/m0101/readme/trouble/niinfo.htm**

Table 1. Sources of driver software

Instrument/ Tools vendor	Finding Driver Availability
Agilent	Agilent drivers are available through the Agilent Developer Network Web site. Go to **http://www.agilent.com/find/ADN** and click on "Downloads." Drivers are listed by type of driver, and by instrument model number.
Vektrex	**http://www.vektrex.com** Tools for developing IVI-COM drivers
Pacific Mindworks	**http://www.pacificmindworks.com/Default.aspx** Tools for developing drivers
Data Translation	**http://www.datx.com/support/** Registration is required to download drivers.
IOtech	**http://www.iotech.com/ftp.html** Listed by instrument type
National Instruments	**http://www.natinst.com/idnet** Allows you to search by instrument vendor, instrument type, etc.
Racal	**http://www.racalinst.com/downloads** After registering on this site, you get a listing by instrument of the types of available drivers.
Tektronix	**http://www.tek.com/site/sw/search/?wt=247&link=/site/sw/search/** Search by product category or model number (drivers co-mingled with software and firmware)
Anritsu	**http://www.us.anritsu.com/downloads/default.aspx?lc=Eng&cc=US&rc=ARO**
Rohde & Schwarz	**http://www.rohde-schwarz.com/** Look under "Shortcuts" for "drivers"

Agilent T&M Programmer's Toolkit (Agilent W1I40A-TK1)

Want to use IVI-C drivers in Visual Studio .NET? Among many other capabilities, Agilent's T&M Programmer's Toolkit (see **www.agilent.com/find/toolkit**) can create managed wrappers around your existing IVI-C and VXI*plug&play* drivers. The wrapper is a native .NET class and fully object-oriented. The T&M Toolkit ships with more than one hundred pre-generated wrappers and its powerful wizard helps you to easily create others. As Figure 11 shows, the Toolkit wizard will:

- Automatically find all your installed drivers
- Find the installed drivers for your instruments, or allow you to download a driver from the Web
- Create a managed wrapper around the raw C-language DLL
- Add the appropriate Project Reference into the project
- Insert sample code to create the driver for your instrument at the proper hardware address

The Wrapper Wizard makes the test engineer's life easier:

- An example of how to call a method on the driver
- IntelliSense help supports all the driver's properties and methods, including help on each method parameter
- Driver call errors are automatically translated into standard .NET exceptions
- Automatic translation of parameters that have only a small range of possible values into a true enumeration, including IntelliSense help on each possible value
- Fully object-oriented implementation of the wrapper makes it intuitive to use

Figure 11. Agilent's IVI-C Driver Wrapper Wizard makes it easy to use IVI-C compliant drivers in VS .NET.

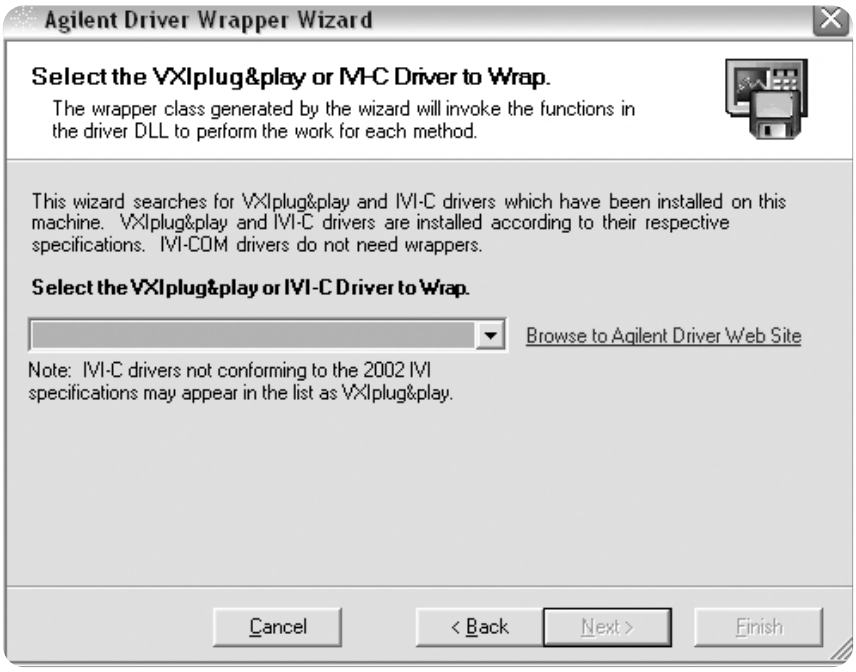
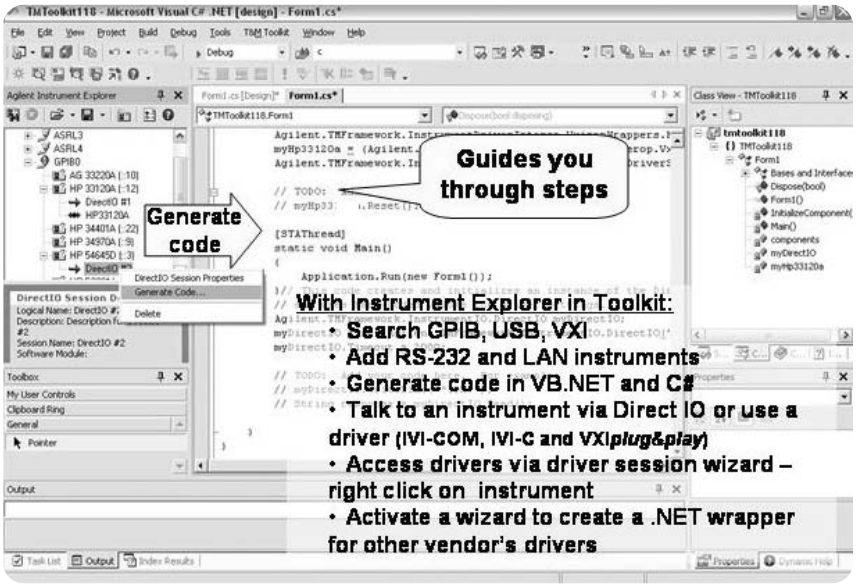


Figure 12. Toolkit saves time. It searches for instruments, talks to them regardless of I/O type, shows all choices for the next function call, writes the VB .NET or C# commands for that function, and gives you context-sensitive help—all in one environment.



Agilent Test Automation Kit (Agilent N1908A)
www.agilent.com/find/kit
The average test system takes 360 hours to configure, test and verify. The Test Automation Kit can save up to 100 of those hours by:

- A USB-to-GPIB converter to simplify installation
- loading all the Instrument Drivers/I/O libraries (included)
- providing a real device and wiring harness for independent verification
- stepping the engineer through the setup process, using Test Express software
- calling any familiar programming language and automatically installing the proper drivers for the instruments present
- a library of over 200 examples in various languages, to use as a head start
- and providing two hours of expert test consulting

Agilent N1908A Test Automation Kit Lit # 5989-0000EN.

Agilent Developer Network
www.agilent.com/find/adn
The Agilent Developer Network is the place to go for

- Drivers
- Downloads
- Discussions

...Instrument Connectivity from Agilent... It simply works.

Glossary

ADE (application development environment) — An integrated suite of software development programs. ADEs may include a text editor, compiler, and debugger, as well as other tools used in creating, maintaining, and debugging application programs. Example: Microsoft Visual Studio.

API (application programming interface) — An API is a well-defined set of set of software routines through which application program can access the functions and services provided by an underlying operating system or library. Example: IVI Drivers

C# (pronounced "C sharp") — new C-like, component-oriented language that eliminates much of the difficulty associated with C/C++.

Direct I/O — commands sent directly to an instrument, without the benefit of, or interference from a driver. SCPI Example: SENSe:VOLTage:RANGe:AUTO

Driver (or device driver) — a collection of functions resident on a computer and used to control a peripheral device.

DLL (dynamic link library) — An executable program or data file bound to an application program and loaded only when needed, thereby reducing memory requirements. The functions or data in a DLL can be simultaneously shared by several applications.

Input/Output (I/O) layer — The software that collects data from and issues commands to peripheral devices. The VISA function library is an example of an I/O layer that allows application programs and drivers to access peripheral instrumentation.

IVI (Interchangeable Virtual Instruments)— a standard instrument driver model defined by the IVI Foundation that enables engineers to exchange instruments made by different manufacturers without rewriting their code.
www.ivifoundation.org.

IVI COM drivers (also known as IVI Component drivers) — IVI COM presents the IVI driver as a COM object in Visual Basic. You get all the intelligence and all the benefits of the development environment because IVI COM does things in a smart way and presents an easier, more consistent way to send commands to an instrument. It is similar across multiple instruments.

Microsoft COM (Component Object Model) — The concept of software components is analogous to that of hardware components: as long as components present the same interface and perform the same functions, they are interchangeable. Software components are the natural extension of DLLs. Microsoft developed the COM standard to allow software manufacturers to create new software components that can be used with an existing application program, without requiring that the application be rebuilt. It is this capability that allows T&M instruments and their COM-based IVI-Component drivers to be interchanged.

.NET Framework — The .NET Framework is an object-oriented API that simplifies application development in a Windows environment. The .NET Framework has two main components: the common language runtime and the .NET Framework class library.

Plug and Play drivers — (also known as universal instrument drivers) are an important category of proprietary drivers. Plug and Play driver standards were originally developed for VXI instruments, and were known as VXI*plug&play* standards. When these standards were adapted for non-VXI instruments they became known simply as "Plug and Play" drivers. Library functions are in accessible C-language source and you can call them from programs written in VEE, BASIC, LabVIEW or LabWindows/CVI.

SCPI (Standard Commands for Programmable Instrumentation) — SCPI defines a standard set of commands to control programmable test and measurement devices in instrumentation systems. Learn more at **http://www.scpiconsortium.org**. See "Direct I/O" for example.

SICL — Standard Instrument Control Library (SICL) is a library of I/O function calls primarily implemented and supported by Agilent. Some of these are core functions that are common across all physical interfaces (GPIB, RS-232, etc.), while others are specific to the interface. The SICL library provides very complete and flexible control of instruments. SICL is optimized for use from C-language and C++ application programs, but can also be used from Visual Basic and other environments that can call arbitrary Windows DLLs. SICL provides complete access to GPIB, RS-232, LAN, VXI message-based, and VXI register-based products.

Universal drivers — another name for Plug and Play drivers

VISA (Virtual Instrument Software Architecture) — The VISA standard was created by the VXI*plug&play* Foundation. Drivers that conform to the VXI*plug&play* standards always perform I/O through the VISA library. Therefore if you are using Plug and Play drivers, you will need the VISA I/O library. The VISA standard was intended to provide a common set of function calls that are similar across physical interfaces. In practice, VISA libraries tend to be specific to the vendor's interface.

VISA-COM — The VISA-COM library is a COM interface for I/O that was developed as a companion to the VISA specification. VISA-COM I/O provides the services of VISA in a COM-based API. VISA-COM includes some higher-level services that are not available in VISA, but in terms of low-level I/O communication capabilities, VISA-COM is a subset of VISA. Agilent VISA-COM is used by its IVI-Component drivers and requires that Agilent VISA also be installed.

VXI*plug&play* — A hardware and software standard that allows interoperability between VXI instruments made by different manufacturers. Learn more at **http://www.vxipnp.org**

Related literature

Data sheets

- *W1140A Software and Connectivity,* pub. no. 5988-5756EN
- *N1908A Test Automation Kit,* pub. no. 5989-0000EN

Application notes

Test-System Development Guide:

- *Introduction to Test-System Design* (AN 1465-1) pub. no. 5988-9747EN **http://cp.literature.agilent.com/litweb/pdf/5988-9747EN.pdf**
- *Computer I/O Considerations* (AN 1465-2) pub. no. 5988-9818EN, **http://cp.literature.agilent.com/litweb/pdf/5988-9818EN.pdf**
- *Understanding Drivers and Direct I/O* (AN 1465-3) pub. no. 5989-0110EN **http://cp.literature.agilent.com/litweb/pdf/5989-0110EN.pdf**
- *Choosing Your Test-System Software Architecture (AN 1465-4)* pub. no. 5988-9819EN **http://cp.literature.agilent.com/litweb/pdf/5988-9819EN.pdf**
- *Choosing Your Test-System Hardware Architecture and Instrumentation* (AN 1465-5) pub. no. 5988-9820EN **http://cp.literature.agilent.com/litweb/pdf/5988-9820EN.pdf**
- *Understanding the Effects of Racking and System Interconnections* (AN 1465-6) pub. no. 5988-9821EN **http://cp.literature.agilent.com/litweb/pdf/5988-9821EN.pdf**
- *Maximizing System Throughput and Optimizing Deployment* (AN 1465-7) pub. no. 5988-9822EN **http://cp.literature.agilent.com/litweb/pdf/5988-9822EN.pdf**
- *Operational Maintenance* (AN 1465-8) pub. no. 5988-9823EN **http://cp.literature.agilent.com/litweb/pdf/5988-9823EN.pdf**
- *Using LAN in Test Systems: The Basics* (AN 1465-9) pub no. 5989-1412EN **http://cp.literature.agilent.com/litweb/pdf/5989-1412EN.pdf**
- *Using LAN in Test Systems: Network Configuration* (AN 1465-10) pub no. 5989-1413EN **http://cp.literature.agilent.com/litweb/pdf/5989-1413EN.pdf**
- *Using LAN in Test Systems: PC Configuration* (AN 1465-11) pub no. 5989-1415EN **http://cp.literature.agilent.com/litweb/pdf/5989-1415EN.pdf**
- *Using USB in the Test and Measurement Environment* (AN 1465-12) pub no. 5989-1417EN **http://cp.literature.agilent.com/litweb/pdf/5989-1417EN.pdf**
- *Using LAN in Test Systems: Applications,* AN 1465-14 (available in February 2005)
- *The IVI Open-Architecture Driver Specifications: An Overview for System Designers,* (AN 1409-4) pub. no. 5988-7939EN

www.agilent.com

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

Agilent Open Connectivity
Agilent simplifies the process of connecting and programming test systems to help engineers design, validate and manufacture electronic products. Agilent's broad range of system-ready instruments, open industry software, PC-standard I/O and global support combine to accelerate test system development. More information is available at **www.agilent.com/find/openconnect.**

By internet, phone, or fax, get assistance with all your test & measurement needs

Online assistance: www.agilent.com/find/assist
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
(fax) (31 20) 547 2390

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

Korea:
(tel) (82 2) 2004 5004
(fax) (82 2) 2004 5115

Latin America:
(tel) (650) 752 5000

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

Other Asia Pacific Countries:
(tel) (65) 6375 8100
(fax) (65) 6836 0252
(e-mail) tm_asia@agilent.com

Printed in the USA December 21, 2004
5989-0110EN

Agilent Technologies