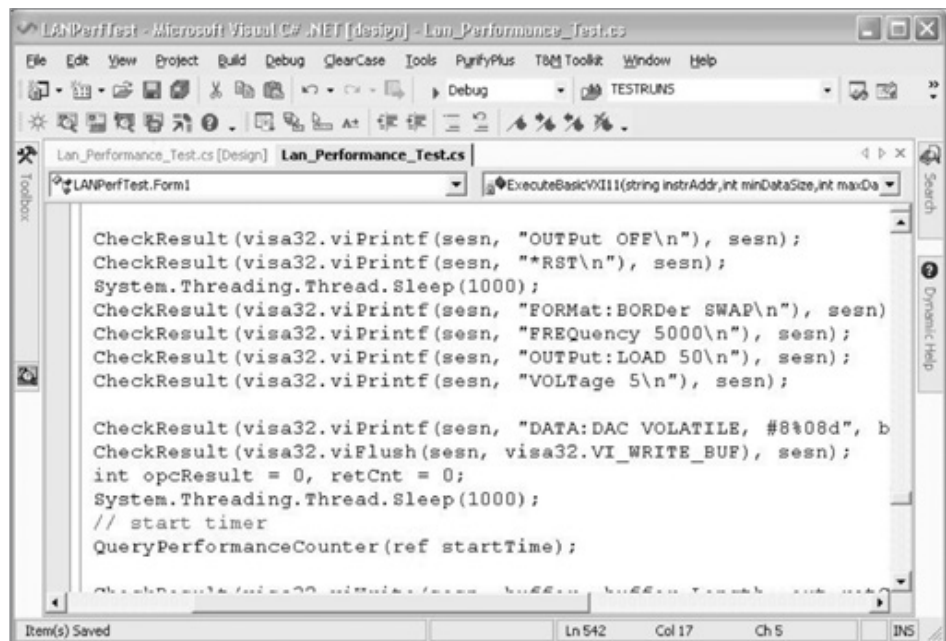


Keysight Technologies

System Developer Guide

Using SCPI and Direct I/O vs. Drivers

Application Note



```
LANPerfTest - Microsoft Visual C# .NET [design] - Lan_Performance_Test.cs
File Edit View Project Build Debug ClearCase Tools PurifyPlus T&M Toolkit Window Help
Debug TESTRUNS
Lan_Performance_Test.cs [Design] Lan_Performance_Test.cs
Lan_PerfTest.Form1 ExecuteBasicVXI11(string instrAddr,int minDataSize,int maxDa
Search Dynamic Help
CheckResult(visa32.viPrintf(sesn, "OUTPut OFF\n"), sesn);
CheckResult(visa32.viPrintf(sesn, "*RST\n"), sesn);
System.Threading.Thread.Sleep(1000);
CheckResult(visa32.viPrintf(sesn, "FORMat:BORDer SWAP\n"), sesn)
CheckResult(visa32.viPrintf(sesn, "FREQuency 5000\n"), sesn);
CheckResult(visa32.viPrintf(sesn, "OUTPut:LOAD 50\n"), sesn);
CheckResult(visa32.viPrintf(sesn, "VOLTAge 5\n"), sesn);

CheckResult(visa32.viPrintf(sesn, "DATA:DAC VOLATILE, #8*08d", b
CheckResult(visa32.viFlush(sesn, visa32.VI_WRITE_BUF), sesn);
int opcResult = 0, retCnt = 0;
System.Threading.Thread.Sleep(1000);
// start timer
QueryPerformanceCounter(ref startTime);
CheckResult(visa32.viPrintf(sesn, "buffin: %s", buffin.ToString());
Item(s) Saved Ln 542 Col 17 Ch 5 INS
```



Overview

This set of application notes shows you how to simplify test system integration by utilizing open connectivity standards such as instrument drivers. The collective goal of these notes is to help you produce reliable results, meet your throughput requirements and stay within your budget.

Using SCPI and Direct I/O vs. Drivers, the fifth note in the series, outlines the relationship between input/output (I/O) software, application software and the ability to maximize instrument interchange and software reuse in present and future systems. This note is a companion to Application Notes 1465-9 through 1465-12, which address the benefits of using a LAN interface in a test system, describe secure topologies for LAN-based test systems, explain how to enable communication between a PC and LAN-enabled instrumentation, and summarize the use of USB in test and measurement. Please see page 14 for a list of the other titles in this series.

Deciding How to Communicate

Once you've chosen an I/O interface for your system—GPIB, LAN, USB or a combination—the next step is deciding how to enable connectivity and achieve communication between the host computer and the instruments in the system. Recently, the alternatives for connectivity and communication have been shifting: vendor-specific commands, libraries and interfaces are giving way to industry-standard command sets, application programming interfaces (APIs) and instrument drivers.

In system development, the use of standards offers two key benefits: it accelerates development by maximizing software reuse and it enhances system flexibility by making it easier to use different instruments. Standards also help you achieve your goals for system functionality and performance by letting you combine methods such as direct I/O with Standardized Commands for Programmable Instruments (SCPI, pronounced “skippy”) and instrument driver-based communication within a single application.

The best choice of I/O software depends on factors such as the number and type of instruments in the system, the functionality to be used within each instrument, the system's throughput requirements, and the number of systems to be deployed. It also depends on which application development environment you're using and the current level of your programming skills.

Sketching the big picture

The diagram in Figure 1 is our starting point. It connects the conceptual side of the discussion—layers of software and hardware—with the actual test system, which includes the computer, I/O cable and test equipment. Within this model, commands and information flow from the application through the software and hardware layers, down the cable, to the instrumentation and back again.

Focusing on the upper-left of the diagram, the application is the program—purchased, downloaded or written by you—that controls the test system. The I/O software layer is the translator that enables communication between the application and the physical I/O hardware—the GPIB, LAN, USB or RS-232 interface in the PC. These three elements reside within the host PC and enable connectivity with the test equipment.

That's all necessary to enable connectivity, but it isn't sufficient to achieve communication. It's similar to the story of placing a phone call to a friend in another country: you pick up the handset, hear the dial tone and dial the number—and then your friend's mother answers the phone. Your inability to speak each other's language prevents meaningful conversation. You have a connection but you haven't achieved communication.

It's the same with test systems. Even if an application has connectivity with an instrument, it must use the right commands and protocols to achieve communication, control, data transfer and so on.

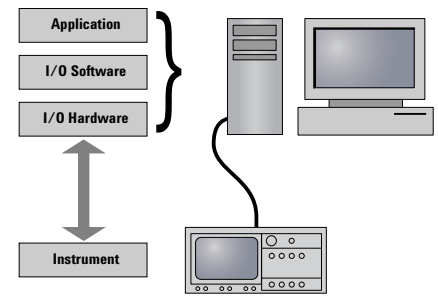


Figure 1. Three essential elements of instrument communication reside within the system's host PC

Enabling Connectivity

In the early days of automated testing, system controllers—called desktop calculators or instrument controllers— had limited processing and sparse memory. To keep the syntax as simple as possible, equipment vendors used short commands, initially in binary and later in ASCII.

Different manufacturers defined their own command strings and these were typically unique to the specific capabilities of each instrument. In a system, replacing an instrument with one from another manufacturer, or even a new-generation product from the original maker, could mean completely rewriting the system software.¹

Instrument commands aren't the whole story. It takes additional layers of software to enable connectivity between a controller and the system instruments. Historically, the I/O software layer contained libraries such as the Standard Instrument Control Library (SICL) or NI-488. The application used these libraries to achieve direct communication with an instrument. Each vendor had a proprietary application programming interface (API) that communicated exclusively with its own I/O interfaces. This made it difficult for system developers who were building mixed-vendor test systems—and, of course, many systems used (and continue to use) equipment from multiple vendors.

Standardizing the API

To make it easier to create mixed-vendor test systems, a group of instrument vendors created the Virtual Instrument Software Architecture (VISA). This provided a standardized API that allowed control of instruments through a common interface—directly or with drivers. From the application's point of view, every vendor's VISA interface looks the same.

One important caveat goes with VISA: Although the VISA API is standard, each vendor employs different layers beneath the VISA layer to control the hardware. In addition, each vendor may have made enhancements to enable unique features in its application layers. To make it all work, the version of VISA installed on the host computer must be compatible with the I/O hardware. (In contrast, this points to another advantage of PC-standard I/O such as LAN and USB: any version of VISA that supports those interfaces will work because the low-level drivers are standardized.)

Expanding freedom of choice

As I/O development was proceeding in the test-and-measurement (T&M) industry, the PC industry was pursuing independence in both I/O and programming languages. Microsoft created the Component Object Model (COM), which is a software architecture that allows components made by different software vendors to be combined into a variety of applications. COM is not dependent on any particular programming language. To incorporate the advantages of language independence, Keysight Technologies, Inc. initiated the creation of VISA COM as a companion to the VISA standard. VISA COM is an object-oriented representation of the VISA API; it exposes the VISA API to the application layer through use of the Component Object Model.

The result: VISA COM gives you the freedom to pick from the most popular I/O configurations and also choose from a wealth of "COM friendly" languages such as C#, Visual Basic 6 and Visual Basic .NET. As we'll discuss later, the application development environment (ADE) you choose will influence the best choice of library and API for your application.

Recapping application notes 1465-9-1465-11

Several factors can increase the burden on the I/O connection in a test system. Examples include the number of instruments in the system, the number of tests being performed, and the volume of commands, status messages and test data being transferred. LAN technology is one of the best ways to handle that burden. It offers a fast, low-cost alternative to GPIB, and it surpasses USB with longer reach and locking connectors.

Most current-generation PCs have built-in LAN ports, which means the computing portion of a test system needs minimal physical configuration. LAN ports are also becoming more common in test equipment. Additionally, devices such as the Keysight E5810A LAN/GPIB gateway make it easy to include older, GPIB-only instruments in LAN-based test systems.

The decision to use LAN for system I/O makes it much easier for colleagues to share data, results, reports and so on. However, it also opens the door to malicious threats and inadvertent risks that can affect system performance and integrity. Fortunately, the creation of a private, protected LAN can shield the test system from many of those risks, and ensure maximum throughput. The standard capabilities of most Windows PCs and many low-cost networking devices enable two viable approaches: one is built around a LAN router and the other is based on a PC equipped with two LAN cards.

A bit more effort is required to create the right environment within a PC for transparent communication with LAN-enabled instruments. Making it work depends on the LAN services of Microsoft Windows XP and several additional capabilities provided by the Keysight IO Libraries Suite 14, which simplifies and accelerates the connection process.

1. For more about the evolution of instrument control, see pages 2-5 of Application Note 1465-3, Understanding Drivers and Direct I/O

Achieving Communication

Once you've enabled connectivity, it's time to decide how to achieve communication between the host computer and the system instrumentation. The two alternatives are direct I/O and instrument drivers. Direct I/O creates an explicit connection to each instrument, which makes it faster but limits instrument interchange and software reuse. Most instrument drivers utilize direct I/O and SCPI but sometimes hide that connection. In all, drivers trade decreased flexibility (and possibly speed) for improved interchange and reusability. However, in most situations you can use both instrument drivers and direct I/O to achieve the best balance of speed, flexibility and measurement functionality.

Standardizing direct I/O

An early attempt at improving consistency and ease of use came in 1989 when Hewlett-Packard introduced an instrument communication language called the Test & Measurement Systems Language (TMSL). HP and eight other manufacturers joined forces to generate a universal approach to instrument control, using TMSL as the starting point. The result was SCPI, the Standard Commands for Programmable Instruments.

The implementation of SCPI within an instrument's firmware has made the programming syntax for direct I/O much more robust and predictable. The syntax defines a strict hierarchy that specifies consistent commands, responses and data formats across instrument models. These commands and responses are defined for source, sense and switch devices. Today, SCPI is still the most-used form of instrument control.

Table 1. This block of Visual Basic 6 code uses SCPI and VISA COM I/O to communicate with a function generator

```

Dim Fgen As VisaComLib.FormattedIO488
' Code removed: Set up the connection to the instrument
With Fgen

    WriteString "*RST"           ' Reset the function generator
    IO.Clear                     ' Clear errors and status registers
    WriteString "FUNCTION PULSE" ' Select pulse waveshape

    WriteString "OUTPUT:LOAD 50" ' Set the load impedance to 50 Ohms (default)
    WriteString "VOLTAGE:LOW 0"  ' Low level = 0 V
    WriteString "VOLTAGE:HIGH 0.75" ' High level = .75 V

    WriteString "PULSE:PERIOD 1e-3" ' 1 ms intervals
    WriteString "PULSE:WIDTH 100e-6" ' Pulse width is 100 us
    WriteString "PULSE:TRANSITION 10e-9" ' Edge time is 10 ns (rise time = fall time)
    WriteString "OUTPUT ON"        ' Turn on the instrument output

For I = 0 To 18                 ' Vary edge by 5 nsec steps
    WriteString "PULSE:TRANSITION " & (0.00000001 + I * 0.000000005)
    Sleep 300                   ' Wait 300 msec
Next I

End With

```

Improving interchange and reuse

SCPI was a big improvement, but the subsequent development of instrument drivers has taken interchange and reuse to new levels. An instrument driver (or just “driver”) is a high-level, instrument-specific (or instrument class-specific) piece of software that enables communication between a PC and an instrument. For software developers, drivers often simplify programming and shorten development time by guiding the programmer through the necessary steps and presenting the capabilities of the instrument within the programming environment (rather than in a manual, as would be the case with SCPI and direct I/O).

First-generation drivers were vendor-specific and typically worked only with a specific ADE. (Numerous legacy application programs still use these proprietary drivers.) Today, however, three types of standardized instrument drivers are available. These work with multiple ADEs and enable communication with an instrument through any vendor’s I/O hardware.

- **VXIplug&play:** Originally developed for modular VXI instruments, these were later expanded to address non-VXI instruments. Conforming drivers always perform I/O through the VISA library. The VXIplug&play WIN32 driver specification works in all popular languages and is today’s most widely used driver architecture.
- **IVI-C:** IVI-C has two distinct drivers. The term is generally applied to drivers based on proprietary tools from NI. With the advent of the IVI standards, NI updated its tools to conform with the standards, but many systems based on the proprietary tools are still in use. To enable reuse and interchangeability, IVI-C requires additional software to patch around its core DLL technology, which does not directly support software interchangeability. An application must call an intermediate driver (an “IVI-C class driver”) which then calls the specific instrument driver to accomplish the function.
- **IVI-COM:** This standard does the most to enable interchangeability and reuse by leveraging the COM computer standard. IVI-COM drivers integrate with standard PC component architecture software and enable control of instruments from familiar, conventional ADEs that provide major productivity improvements. IVI-COM drivers that control VXI or GPIB instruments use VISA (either VISA COM or VISA-C). Because many new instruments include computer-standard I/O such as LAN and USB, IVI-COM drivers for non-GPIB instruments are not required to use VISA, although many do.

If you are unsure of which I/O technology an application or driver is using, take a look at the connection string or “instrument address” used for instrument communication. VISA-type strings look like “TCPIP: 34980A.tm.keysight.com::inst0::INSTR” while SICL-based strings are similar to “lan[34980A.tm.keysight.com]:inst0.”

Exploring the Application Alternatives

Shrink-wrapped software often provides convenience in measurement and analysis at the expense of performance and flexibility. Such products are often a good fit with the small or one-off systems used during product development. In contrast, custom-built software is often the best answer for applications such as design verification or manufacturing test that require high performance and maximum flexibility.

Simplifying basic analysis tasks

There are alternatives to general-purpose development environments. One example is “targeted applications,” which address specific measurement or technology domains, or specific phases or tasks in the product development lifecycle. These applications include software designed to make the infrequent measurements (manual or semi-automated) that are typically performed during the early phases of product development or during design verification.

Applications such as Keysight IntuiLink connectivity software (free) and Keysight BenchLink (low cost) make it easy to perform semi-automated measurements, collect data, and analyze results from a wide variety of instruments. Both applications utilize either drivers or direct I/O—transparently—to enable instrument communication, control and data transfer.

- **IntuiLink:** This connectivity application simplifies data transfers by adding a toolbar to popular PC applications such as Microsoft Word and Excel. IntuiLink enables direct retrieval of data and images from a measurement instrument, letting you remain in the PC application and use its familiar interface. IntuiLink also eliminates barriers between instruments and PCs by supporting GPIB, USB, LAN and FireWire interfaces.
- **BenchLink:** This low-cost application is available in versions that support numerous instruments. BenchLink is a Windows®-based application (Figure 2) that uses a familiar spreadsheet format to streamline data collection, presentation and analysis. It can communicate with measurement instruments via LAN, USB or GPIB using the included I/O software.

There are higher-cost alternatives to BenchLink, including instrument-control software for functional testing and domain-specific applications. These range from general test executives to application-specific programs such as cell phone regulatory testing tools. All serve to further reduce the burden of instrument programming, connectivity and communication.

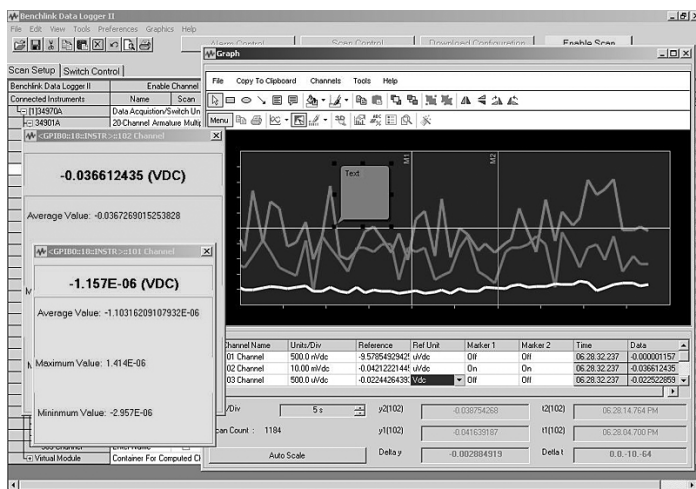


Figure 2. Keysight BenchLink Data Logger provides spreadsheet-like test set up and real-time display and analysis of measurements

Comparing development environments

The software environment you choose will have a significant impact on the time, effort and cost required to create and maintain a test system. Development environments are either graphical or textual. Graphical environments such as Keysight VEE Pro and NI LabVIEW use a schematic approach, which is regarded as being easy for engineers to learn. You manipulate icons or objects that represent commands or functions and connect them with program-flow lines. This makes it easier to visualize the paths of execution and interaction; it also shields you from the underlying syntax. What’s more, T&M-specific graphical environments have extensive I/O and instrument drivers as well as measurement-related math and graphing capabilities. Graphical programming is best suited to small- and medium-sized applications—the visual interface tends to become difficult to understand with large programs.

In contrast, textual programming has a steeper learning curve because it requires detailed knowledge of a language’s commands and syntax. However, because most textual languages are based on open standards, they offer a greater selection of development environments, software tools and training opportunities. There also tends to be a wider variety of available third-party drivers, tools and add-ons. Textual programming is often the best choice for large, comprehensive programs because it is easier to navigate and comprehend.

In the past, textual programming produced applications that had pronounced speed advantages—at runtime—over those created with graphical programming. Today, however, there is less difference in runtime speeds between applications created with either approach.

Maximizing Performance and Flexibility

You can pick from a wide variety of alternatives that support the creation of custom measurement software. These range from test automation applications to full-featured development environments that utilize either graphical or textual programming. Your preferred approach will determine the best choice for instrument communication.

Microsoft Visual Studio

Visual Studio is a textual programming solution that offers an extensive range of developer tools and built-in help capabilities that can accelerate development of Windows-based applications. Its integrated development environment provides a consistent interface for all supported languages, including Visual Basic, C++ and C#. As a standardized, mainstream development product, Visual Studio offers several advantages:

- **Open:** Because Visual Studio is based on open, pervasive standards, it can communicate with practically any other programming technology. As a result, thousands of third-party tools—software, drivers, etc.—are available to support your development efforts.
- **COM-friendly:** Visual Studio works very well with programming technologies that are based on Microsoft's COM technology. This includes VISA COM and IVI-COM.
- **On-screen help:** The IntelliSense feature and the "F1 help" capability work with COM- and .NET-based third-party drivers and software. As an example, the IntelliSense window for a driver will show all available operations, a brief description of each, and a summary and description of all allowed parameters. Depending on the driver or component, pressing the F1 key may open a new window that presents an online help manual for the driver. Using this type of on-screen, context-sensitive help is much faster than thumbing through a printed programming manual.

There is one downside in test-system applications: it can be difficult to use C APIs with the new .NET-based languages in Visual Studio. The latest releases of Microsoft programming languages utilize .NET technology to communicate with drivers and third-party software—and .NET is rapidly phasing out C API technology. This affects the C API version of the VISA I/O library as well as IVI-C and VXIplug&play drivers. To get around this problem, Keysight provides a .NET wrapper for the VISA API. The wrapper is available as a free download from www.Keysight.com/find/iolib; it is also included in the Keysight IO Libraries product.

Visual Studio with Keysight T&M Toolkit

The Keysight T&M Toolkit 2.0 with test automation extends the .NET-enabled versions of Visual Studio with a suite of integrated, easy-to-use software tools and components—project wizards, APIs, class libraries, widgets, graphs, drivers and more. This creates an environment that simplifies the process of incorporating tests and measurements into custom applications. Using T&M Toolkit 2.0 within the Visual Studio environment lets you use your preferred textual programming language and integrate your new code with existing code written in other languages.

T&M Toolkit 2.0 offers several other capabilities that speed and simplify system development:

- **DirectIO class:** This is the easiest way to send commands directly to an instrument.
- **Wrapped VXIplug&play drivers:** This integrates the drivers into .NET with full IntelliSense and F1 help capabilities. T&M Toolkit also recognizes and uses IVI-COM drivers, which have IntelliSense built-in.
- **Instrument Explorer:** This tool makes it easy to see and edit the instrument I/O configuration and initiate communication with instruments.
- **IO Monitor:** This utility makes it much easier to use instrument-control software and instrument drivers—IVI-COM, VXIplug&play—and diagnose problems by letting you watch both the underlying direct I/O commands that are sent to the instrument and the resulting data that is returned (Figure 3).

In all, the combination of Visual Studio and T&M Toolkit eliminates many of the difficulties often associated with connecting to and controlling test equipment from a custom application.

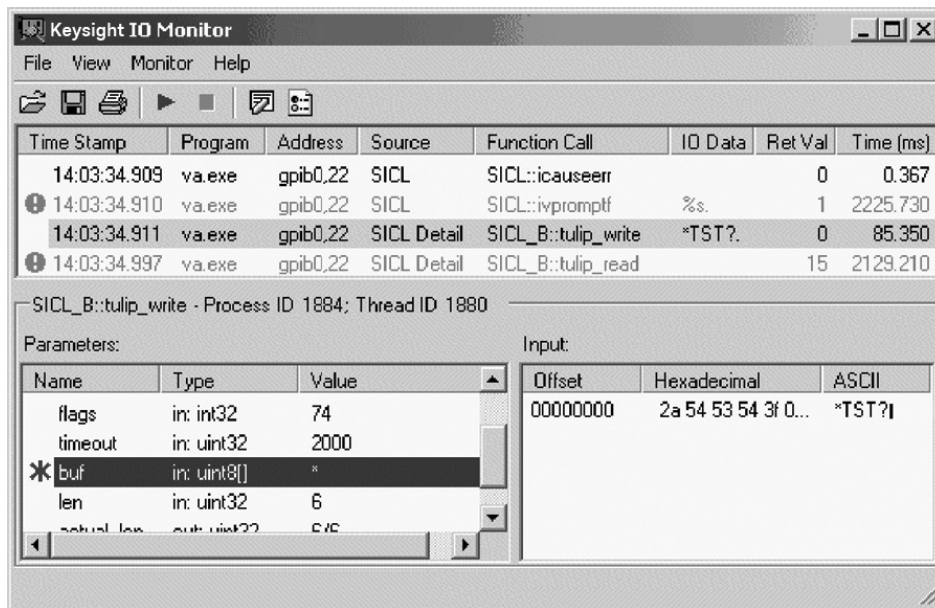


Figure 3. T&M Toolkit's IO Monitor traces I/O layers for Keysight's VISA, VISA COM, SICL and SICL Detail, helping you find bottlenecks in your source code

Keysight VEE Pro

For those who want an alternative to textual programming, Keysight VEE Pro 7.0 is a powerful, easy-to-use graphical programming environment that accelerates the process of building and programming test systems. To create a program, you choose high-level graphical objects from a huge library and connect them with lines or “wires.” The wire connections specify functionality and sequences within intuitive block diagrams.

Because VEE Pro is an open, standards-friendly environment, it also offers several advantages in test-system development:

- **Direct I/O:** Through its easy and powerful Direct I/O capability, VEE Pro provides excellent support of direct I/O for control of any standard instrument and many vendors’ PC plug-in cards.
- **Instrument drivers:** VEE Pro supports industry-standard drivers such as IVI-COM and VXIplug&play. It includes support for nearly one thousand drivers, supporting popular instruments from more than 70 different manufacturers.
- **COM and .NET:** No familiarity with .NET programming languages is required to utilize these capabilities. VEE Pro takes care of the details, ensuring successful interaction with both COM and .NET software.

Assessing I/O Software Alternatives

Our ultimate goal is to minimize the amount of time you have to spend sorting out which I/O libraries or drivers to use in your test systems. Today, however, that effort is unavoidable—but we can offer a few suggestions that will simplify the process.

Instrument drivers vs. direct I/O

When comparing drivers and direct I/O, there are two key factors to consider. One is a tradeoff between speed of development and speed of execution: drivers contribute to faster development while direct I/O enables faster execution.

The other factor is access to instrument functionality. Drivers typically cover a subset of an instrument's total feature set—and this is often limited to the most commonly used functions. In contrast, the combination of direct I/O and SCPI commands can typically access 100 percent of an instrument's programmable functions, no matter how arcane. If you prefer the advantages of drivers but need to access unsupported features, it is possible to use both methods within an application.

ADE vs. I/O API

The ADE you select will affect the best choice of I/O library and API for your application. Table 2 shows the various I/O APIs that Keysight supports and, for each ADE, highlights the recommended library as well as the preferred and historical alternatives.

As one noteworthy example, we recommend VISA COM over the VISA API when using Visual Basic 6 because VISA COM is an object-oriented, hierarchical view of the VISA API. Using the COM version means you don't have to add the .bas file to the VB project (though the reference is needed) and VISA COM allows for the use of context-sensitive IntelliSense help.

Table 2. ADEs and recommended I/O libraries

Application development environment	Recommended library	Supported alternatives	
		Preferred	Historical
Visual Basic 6	VISA COM	VISA with visa32.bas	SICL
Visual C/C++	VISA with visa32.h	VISA COM	SICL
Visual Basic .NET, C# and other .NET languages	T&M Toolkit DirectIO	VISA COM VISA with visa32.cs VISA with visa32.vb	

ADE vs. instrument driver

As mentioned earlier, three types of standardized instrument drivers are available: VXIplug&play, IVI-C and IVI-COM. These work with multiple ADEs and enable communication with an instrument through any vendor's I/O hardware.

Reading from left to right, Table 3 shows a continuum that ranges from least to most standardized across three generations of drivers—proprietary, T&M standard and PC-industry standard. These represent the past, present and future of driver technology.

To accelerate test-system development, we recommend using the latest IVI-COM drivers and VXIplug&play WIN32 drivers for instrument control. The IVI-COM driver technology is the only one built on a PC-standard architecture. A component driver built on COM works in all popular PC languages and most T&M languages. What's more, it utilizes the most popular types of I/O and can be used in the latest .NET technologies.

Table 3. ADEs and their recommended instrument drives

Instrument driver families		
Proprietary T&M (specific to one language)	Test & Measurement (based on T&M standards)	Component PC (based on PC standards)
<ul style="list-style-type: none"> · LabVIEW Plug & Play (VXIplug&play GWIN) · VEE Panel Drivers 	<ul style="list-style-type: none"> · LabWindows/CVI Plug & Play · WIN VXIplug&play · IVI-C 	<ul style="list-style-type: none"> · IVI-COM

Shaping The Future of Test Systems

Open standards such as COM and LAN have achieved widespread adoption in the computer world and are now shaping the future of test-system development. Standards accelerate system development by maximizing software reuse and enhance system flexibility by making it easier to swap out instruments—different models and even different brands. Standards also enhance system functionality and performance by letting you utilize direct I/O, SCPI and drivers within a single application.

Your choice of development environment can make it easier to incorporate tests and measurements into custom applications. If you prefer textual programming, Visual Studio with Keysight T&M Toolkit eliminates many of the problems associated with connecting to and controlling test equipment. If you prefer graphical programming, Keysight VEE Pro is an open, standards-friendly environment that supports direct I/O and instrument drivers as well as COM and .NET technologies.

To discover more ways to simplify system integration, accelerate system development and apply the advantages of open connectivity, please visit the Web site at www.keysight.com/find/open. Once you're there, you can also connect with our online community of system developers and sign up for early delivery of future application notes in this series. Just look for the link "Join your peers in simplifying test-system integration."

Glossary

- ADE** – application development environment; an integrated suite of software development programs that may include a text editor, compiler and debugger as well as other tools used to create, maintain and debug application programs
- API** – application programming interface; a well-defined set of software routines through which an application program can access the functions and services provided by an underlying operating system or a reusable software library
- C#** – pronounced “C sharp;” a recent component-oriented programming language that resembles C++ and combines attributes of the C++ and Java languages
- COM** – Common Object Model; also called Microsoft COM; allows software developers to create new software components that can be used with an existing application program without modifying the program; an improvement over DLLs for software reuse
- Direct I/O** – direct input/output; enables communication with an instrument without benefit (or overhead) of a driver; successful use of direct I/O typically requires a strong understanding of Standard Commands for Programmable Instrumentation (SCPI)
- DLL** – dynamically linked library; a set of software operations used by other application programs; can be loaded at any time and can serve as a container for a reusable software library that can be shared simultaneously by multiple applications
- Driver** – also called an instrument driver; a collection of functions resident on a computer and used to control an instrument (e.g., DMM, oscilloscope, network analyzer); an alternative to SICL, direct I/O and VISA
- GPIB** – General Purpose Interface Bus; the dominant 8-bit parallel I/O connection for test equipment and test systems
- HP-IB** – Hewlett-Packard Interface Bus; another name for GPIB
- Input/output layer** – also called the I/O layer; the software that interacts with peripheral devices (e.g., instruments), issuing commands and collecting data
- IVI** – Interchangeable Virtual Instruments; a standard instrument driver model that allows a consistent programming style across instrument models and classes
- IVI-COM drivers** – also called IVI component drivers; presents the IVI driver as a COM object, preserving the full capabilities of your preferred development environment
- LAN** – local area network
- Library** – a collection callable software operations; reusable software functions meant to be used by other programs
- .NET Framework** – a platform for application development that provides a large library of operations, encourages software reuse, reduces programmer error and simplifies application development in a Windows environment; its two main components are the common language runtime and the class libraries
- Plug and Play drivers** – also called universal instrument drivers; an adaptation of VXIplug&play drivers for non-VXI instrumentation; library functions that can be called from user-written programs
- SCPI** – Standard Commands for Programmable Instrumentation; defines a universal set of commands for control of programmable test equipment
- SICL** – Standard Instrument Control Library; a modular instrument communications library that works with a variety of computer architectures, I/O interfaces and operating systems; superseded by VISA
- USB** – Universal Serial Bus; designed to replace the RS-232 and RS-422 serial buses used in PCs
- UPnP** – Universal Plug and Play; a networking architecture that ensures compatibility of devices, software and peripherals; not the same as Plug and Play or VXIplug&play drivers
- VISA** – Virtual Instrument Software Architecture; sometimes called VISA-C; a common foundation for system software components, including instrument drivers, virtual front panels and application software; consists of a vendor-independent set of instrument communication operations that work across different I/O interface technologies
- VISA COM** – provides the services of VISA in a COM-based API; a subset of VISA in terms of I/O capabilities but includes some services not available in VISA
- VXI** – VME extensions for instrumentation; a standard, open architecture for modular test instrumentation and systems
- VXIplug&play** – the most popular driver technology for all types of instrumentation; provides a consistent programming style across instruments; includes virtual front panel technology that allows development environments to provide extra help and visual guidance for operating an instrument
- Wrap** – the process of adding interface software that creates compatibility between the enclosed (wrapped) software module and other programs or modules
- Wrapper** – additional software that acts as an interface between an enclosed (wrapped) software module and other programs or modules

Related Literature

The other notes in this series provide additional information about the successful use of LAN in test systems:

- Using LAN in Test Systems: The Basics, AN 1465-9 (pub no. 5989-1412EN)
<http://cp.literature.keysight.com/litweb/pdf/5989-1412EN.pdf>
- Using LAN in Test Systems: Network Configuration, AN 1465-10 (pub no. 5989-1413EN)
<http://cp.literature.keysight.com/litweb/pdf/5989-1413EN.pdf>
- Using LAN in Test Systems: PC Configuration, AN 1465-11 (pub no. 5989-1415EN)
<http://cp.literature.keysight.com/litweb/pdf/5989-1415EN.pdf>
- Using USB in the Test and Measurement Environment, AN 1465-12 (pub no. 5989-1417EN)
<http://cp.literature.keysight.com/litweb/pdf/5989-1417EN.pdf>
- Using LAN in Test Systems: Applications, AN 1465-14 (available in March 2005)

Other Keysight application notes provide additional hints that can help you develop effective test systems:

- Creating a Wireless LAN Connection to a Measurement System (AN 1409-3) pub no. 5988-7688EN
<http://cp.literature.keysight.com/litweb/pdf/5988-7688EN.pdf>
- Introduction to Test System Design (AN 1465-1) pub no. 5988-9747EN
<http://cp.literature.keysight.com/litweb/pdf/5988-9747EN.pdf>
- Computer I/O Considerations (AN 1465-2) pub no. 5988-9818EN
<http://cp.literature.keysight.com/litweb/pdf/5988-9818EN.pdf>
- Understanding Drivers and Direct I/O (AN 1465-3) pub no. 5989-0110EN
<http://cp.literature.keysight.com/litweb/pdf/5989-0110EN.pdf>
- Choosing Your Test-System Software Architecture (AN 1465-4) pub no. 5988-9819EN
<http://cp.literature.keysight.com/litweb/pdf/5988-9819EN.pdf>
- Choosing Your Test-System Hardware Architecture and Instrumentation (AN 1465-5) pub no. 5988-9820EN
<http://cp.literature.keysight.com/litweb/pdf/5988-9820EN.pdf>
- Understanding the Effects of Racking and System Interconnections (AN 1465-6) pub no. 5988-9821EN
<http://cp.literature.keysight.com/litweb/pdf/5988-9821EN.pdf>
- Maximizing System Throughput and Optimizing System Deployment (AN 1465-7) pub no. 5988-9822EN
<http://cp.literature.keysight.com/litweb/pdf/5988-9822EN>
- Operational Maintenance (AN 1465-8) pub no. 5988-9823EN
<http://cp.literature.keysight.com/litweb/pdf/5988-9823EN>

This document was formerly known as application note 1465-13.

ATCA®, AdvancedTCA®, and the ATCA logo are registered US trademarks of the PCI Industrial Computer Manufacturers Group

www.keysight.com/find/5989-1414EN.pdf

