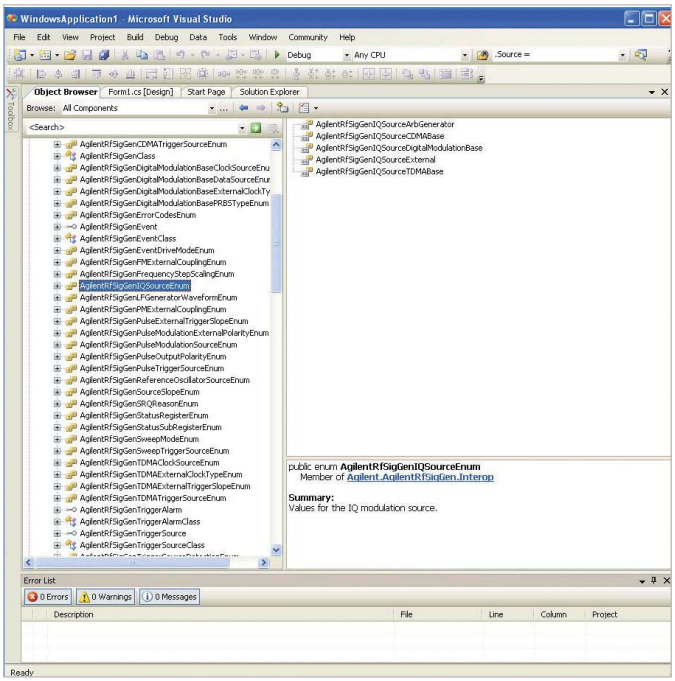
Assessing the use of IVI drivers in your test system: Determining when IVI is the right choice

Application Note



If software reuse and system "portability" are important to your organization, it may be to your advantage to use IVI drivers in present and future test systems. This is especially true if you are making the shift from GPIB-based control to LAN and LXI: All LXI-compliant instruments include an IVI driver.

To help you decide if IVI drivers are right for your system, this application note provides an overview of the technology, presents additional considerations, and describes the tradeoffs. The note includes eight major sections:

- Choices in instrument control
- Why use an instrument driver?
- Benefits of using an IVI driver
- IVI driver classes
- Requirements to use an IVI driver
- Tradeoffs and limitations of IVI drivers
- Steps to getting started
- Using IVI drivers in Visual Basic .NET or Visual C#

Much of the material in this application note has been leveraged from the IVI Foundation's *IVI Getting Started Guide*. Sections taken directly from the *Getting Started Guide* are marked in *italics*. That document is available as a PDF from the IVI Foundation Web site at **www.ivifoundation.org**.

Agilent Technologies

Choices in instrument control

There are four choices available for providing computer-based control of most instruments:

- Standard Commands for Programmable Instrumentation (SCPI)
- A program- or language-specific driver (e.g., a LabVIEW driver)
- A proprietary driver (supplied with many vendors' instruments)
- An open driver such as IVI

Each of these carries a set of tradeoffs. For example, choosing a proprietary or program-specific driver may result in very tight integration with any provided tools or programs; however, this comes at the expense of test system portability. You may also be confronted with a common problem: the available set of instrument drivers doesn't match your desired (or required) list of instruments. We'll talk about the tradeoffs in using an IVI driver in a later section.

SCPI control is one of the most popular methods of programmatically controlling instruments. These commands work on instruments that have implemented an appropriate SCPI command set and also contain an on-board command parser, typically implemented through a microprocessor or state machine. This specifically means that SCPI control will not work on most register-based devices housed in a cardcage (e.g., neither VXI nor PXI architectures).

SCPI provides fine-level instrument control through a consistent, test-focused command set. However, it can be complicated to use with more complex instruments where it may seem like programming a computer in assembly language. SCPI carries three other important considerations:

- It is not dependent on the operating system or programming language so is highly portable among test systems.
- It defines common functions, so is generally portable among instruments.
- It does not require a certain set of functions, so some calls may not be implemented on similar instruments.

In many cases, SCPI is a suitable and sufficient choice. However, drivers provide extended capabilities that simplify instrument control and reduce programming time.

A quick look at SCPI

Beginning in 1989, a group of instrument manufacturers developed Standard Commands for Programmable Instrumentation (SCPI). This defined a set of commands for controlling instruments using ASCII characters, providing some basic standardization and consistency to the commands used to control instruments. For example, when you want to measure a DC voltage, the standard SCPI command is "MEASURE:VOLTAGE:DC?."

Why use an instrument driver?

To understand the benefits of IVI drivers, we need to start by defining instrument drivers in general and describing why they are useful. An instrument driver is a set of software routines that controls a programmable instrument. Each routine corresponds to a programmatic operation such as configuring, writing to, reading from, or triggering the instrument. Instrument drivers simplify instrument control and reduce test program development time by eliminating the need to learn the programming protocol for each instrument.

Although you must be syntactically correct in all calls to an instrument driver, making calls to a subroutine in your preferred programming language is less error prone. This is especially true if you are using a language that supports the *Intellisense* capability, which presents valid syntax choices as you type each command. An instrument driver such as IVI integrates more tightly with your programming language than will SCPI calls through the VISA I/O library.

If you've been programming instruments without using a driver, then you are probably well acquainted with the chore of hunting through the programming guide while looking for the right SCPI command and its exact syntax. You've also been dealing with an I/O library to format and send strings, and then embedding response strings into variables. Let's take a look at some advantages of IVI drivers.

Benefits of using an IVI driver

Before IVI drivers, there was a standard framework called VXI*plug&play* drivers. This specification defined a standard dynamic-link library (.dll) interface; however, did not provide for interchangeability among similar types of instruments. In fact, the normal syntax of a VXI*plug&play* driver discouraged inter-vendor operability because the keywords contained vendor names. Furthermore, the standard did not define standard keywords for common actions. For instance, one vendor may use "read" while another may use "get" as the syntax for acquiring a measured value from an instrument.

In contrast, the IVI standard defines the concept of an "instrument class." For any type or "class" of instrument, common functions have been defined. Using digital multimeters (DMMs) as an example, the IVI driver for every DMM from any vendor uses the measurement command IviDmmMeasurement.Read. Once you learn to program the IVI-specified commands for the instrument class, you can use any vendor's instrument without relearning the commands. Commands that are common to all drivers, such as *Initialize* and *Close*, are identical for every instrument class. This commonality ensures portability across test systems and lets you spend less time searching through help files or programming an instrument—and leaves more time for your primary job.

This level of commonality—or interoperability—was the motivation behind the development of IVI drivers. The specifications define an open driver architecture, a set of instrument classes, and shared software components. Together these provide consistency and ease of use, as well as the crucial elements needed for advanced features. IVI drivers also support instrument simulation, automatic range checking and state caching. These capabilities provide several benefits:

- Instrument simulation enables program development and debugging even when an instrument is not available. This is particularly useful when you are waiting for the test hardware to arrive or if several people must share expensive test resources.
- Automatic range checking can eliminate unnecessary programming errors.
- State caching is optionally implemented within the driver. This capability can accelerate the execution of certain test programs by making only the minimum changes to the state of an instrument.

Eliminating unnecessary instrument resets, range changes or other similar functions can further accelerate the execution of measurement functions.

IVI driver classes

The IVI Foundation has created IVI class specifications that define a common set of capabilities for drivers for eight instrument classes (Table 1).

Class	IVI driver
Digital multimeter	IviDmm
Oscilloscope	IviScope
Arbitrary waveform/ function generator	IviFgen
DC power supply	IviDCPwr
Switch	IviSwitch
Power meter	IviPwrMeter
Spectrum analyzer	IviSpecAn
RF signal generator	IviRFSigGen

Table 1. The IVI standard includes eight instrument classes.

In addition to the existing class drivers, there are six other classes in development as this paper is being written. The timer/counter and AC power classes are directed at general-purpose instruments while upconverter, downconverter, digitizer, and arb (baseband) classes are being developed as part of the synthetic instruments program spearheaded by the US military.

It is common for instruments within a specific class to provide IVI class-compliant drivers to additionally provide functions beyond the defined IVI Class definition. There may be two reasons for this: Either the capability is not common to all instruments of the class or because the instrument offers a level of control that is more refined than what the class defines.

IVI defines "custom drivers," which are used for instruments that are not members of a class. For example, there is not a class definition for network analyzers, so those drivers must be custom. Custom drivers provide the same consistency and benefits described above for an IVI driver, but are specific to a device so lack interchangeability across vendors.

Requirements to use an IVI driver

The primary gating item is the availability of a specific IVI driver for the instrument you want to control. If such a driver is available, it must be installed on the controlling computer along with four other essential items:

- An appropriate Microsoft® operating system and certain operating-system services.
- An appropriate programming language.
- An appropriate I/O library.
- IVI shared components code.

With these items all in place, you are ready to proceed.

Tradeoffs and limitations of IVI drivers

The various companies that defined the IVI standard tried to minimize the tradeoffs in using an IVI driver. Even so, there five key considerations you should keep in mind:

- An IVI driver may or may not take full advantage of the speed capabilities of an instrument.
- An IVI driver will seldom implement the entire feature set of an instrument.
- IVI drivers are not available for all commercially available instruments.
- The two types of IVI drivers, IVI-C and IVI-COM (Common Object Model) (see sidebar), mean that you will need to choose one or the other. To make an informed choice, it's useful to understand the benefits and tradeoffs of each type.
- Your system must meet the requirements outlined in the previous section

Let's expand on the first three points.

Types of IVI drivers

To support all popular programming languages and development environments, IVI drivers provide either an IVI-C or an IVI-COM application programming interface (API). Driver developers may provide either or both interfaces, as well as wrapper interfaces optimized for specific development environments.

Although the functionality is similar, IVI-C drivers are optimized for use in ANSI C development environments while IVI-COM drivers are optimized for environments that support COM. IVI-C drivers extend the VXI*plug&play* driver specification and their usage is similar. IVI-COM drivers provide easy access to instrument functionality through methods and properties. IVI-C drivers may require more active management of driver versions to ensure compatibility.

All IVI drivers communicate with the instrument through an I/O library. We recommend using the Virtual Instrument Software Architecture (VISA) library, a widely used standard library for communicating with instruments from a personal computer.

An IVI driver may or may not take full advantage of speed capabilities: Instrument drivers are generally written for a general user and may include extra code such as range checking to ensure proper operation. If you are operating an instrument within well-known parameters, you can often program a more concise set of instrument commands than those provided by the driver. Additionally, many drivers are built to maximize programming compatibility across interfaces (e.g., GPIB, LAN and USB) and may use compatibility modes that do not take full advantage of instrument speed. On the other hand, some drivers incorporate state caching, which eliminates unnecessary steps and accelerates operation. As a result, it is very difficult to know if a driver will provide the fastest possible execution. Often, advanced programmers will use a driver for most instrument functions, then will selectively tune their code (e.g., by using SCPI calls or direct register operations) for time-critical operations.

An IVI driver may not implement the entire feature set: A simple DMM may have only 25 commands, whereas 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. That's why you'll seldom see a driver that covers every command in an instrument. Instrument manufacturers take their best guess at the commands you are likely to use and craft the driver accordingly. For most users, the commands available in the driver will cover their programming needs. Advanced users may be disappointed in the functionality of some drivers and might consider other alternatives.

IVI drivers are not available for all instruments: It isn't practical for instrument manufacturers to create drivers for all past models. IVI drivers are available for most new instruments and many popular older ones, but you should verify driver availability before beginning a project.

Getting started

The following 10 steps will help you prepare your system using IVI instrument drivers.

1. Choose a programming language. We'll show examples for VB .NET and Visual C. For other languages, see the *Getting Started Guide* available from the IVI Foundation Web site. **www.ivifoundation.org/**
2. Install the I/O libraries. Agilent I/O libraries are available at no charge to Agilent customers. See **www.agilent.com/find/iolib** to download the latest version.
3. Download and install the IVI shared components. These are available from vendor Web sites or directly from the IVI Foundation Web site
4. Download and install the appropriate IVI driver for any instruments you will be controlling. These are commonly available at the vendor's Web site. For Agilent instruments, see **www.agilent.com/find/drivers**. The IVI Foundation Web site also has links to many instrument vendor sites. Note that occasionally drivers will unload into multiple files. When you reference the driver (see below), you should choose the files you need based on the functionality you require.
5. Determine the VISA address string.
6. Reference the driver or load driver files.
7. Create an instance of the driver if your development environment is using COM.
8. Create a variable to represent your instrument(s).
9. Initialize the instrument(s).
10. Write your test program. Be sure any instruments are closed before exiting your program as this frees any system resources reserved for those devices.

Using IVI drivers in Visual Basic .NET or Visual C#

Let's start with Microsoft Visual Studio®, which is a popular development environment. The procedures for Visual C# and Visual Basic .NET are very similar. Start by installing the required software from the list above: operating system, Visual Studio with appropriate compiler, I/O libraries, IVI shared components and any IVI instrument drivers. We recommend using the IVI-COM driver in Visual Studio whenever available.

To use a driver, you must first add a reference to it. After that, the following steps are recommended:

1. *Launch Visual Studio and start a new Console Application (in Visual C# or Basic). Keep any code generated.*

2. *Select <Project> and <Add Reference>. The Add Reference dialog box will appear.*

3. *Select the COM tab. Scroll to the IVI section (all IVI drivers begin with IVI) and select the type library for your chosen instrument(s). Any IVI drivers you have installed should appear in this list. Select the driver(s) and press <OK>.* Note that you can add multiple drivers with one selection. Note also that a single instrument will sometimes contain more than one driver. See Figure 1 as an example.

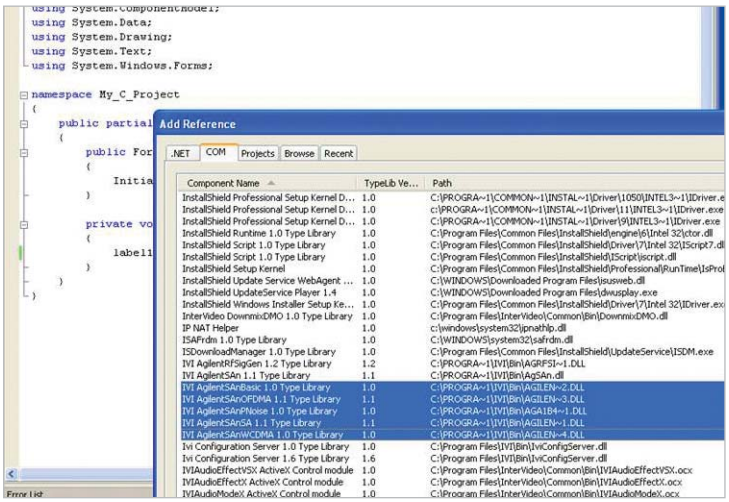


Figure 1. A single instrument may include more than one driver.

4. Create an instance of each driver by adding a *using* statement to allow your program to access the driver. For an Agilent driver, type "using A" and *Intellisense* will show the available Agilent drivers (e.g., *using Agilent.A* will pop up the list of installed drivers). Put this directly beneath the other using statements in your code. See Figure 2 for an example of the *Intellisense* popup.

5. Initialize the instrument: Create a variable to represent your instrument and set the initialization parameters.

   For Visual Basic.NET:

   a. Enforce type checking (Option Explicit On).

   b. Add Dim variables for the instrument and for the reading.

6. *To view the functions and parameters available in the instrument driver, right-click the library in the References folder in Solution Explorer and select View in Object Browser.*

7. You can now start writing the program to use the available instrument drivers.

8. When you are finished, be sure to close the session.

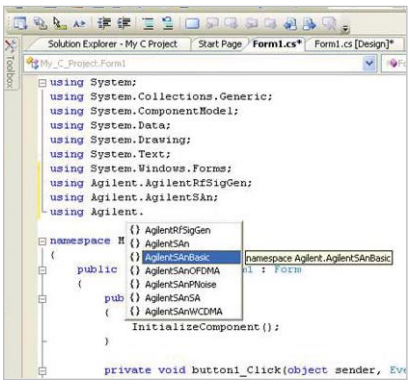


Figure 2. Depending on the context, the Intellisense *popup window will show available drivers or installed drivers.*

Conclusion

The optimum choice of instrument control method depends on many factors. Today, IVI drivers are being shipped with more and more new instruments—so it may be time to consider using them in your application. The increasing availability of IVI drivers along with their tight integration with modern programming languages means that IVI drivers could be the best choice for your next test programming project.

www.agilent.com
www.agilent.com/find/XXX

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.

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. For information regarding self maintenance of this product, please contact your Agilent office.

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

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

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

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	0120 (421) 345
Korea	080 769 0800
Malaysia	1 800 888 848
Singapore	1 800 375 8100
Taiwan	0800 047 866
Thailand	1 800 226 008

Europe & Middle East

Austria	01 36027 71571
Belgium	32 (0) 2 404 93 40
Denmark	45 70 13 15 15
Finland	358 (0) 10 855 2100
France	0825 010 700*
	*0.125 €/minute
Germany	07031 464 6333
Ireland	1890 924 204
Israel	972-3-9288-504/544
Italy	39 02 92 60 8484
Netherlands	31 (0) 20 547 2111
Spain	34 (91) 631 3300
Sweden	0200-88 22 55
Switzerland	0800 80 53 53
United Kingdom	44 (0) 118 9276201

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

Revised: October 1, 2008

Printed in USA, November 14, 2008
5990-3186EN

Agilent Technologies