Selecting the Right Device Driver for PXI Hardware – VISA or IVI?

By Alan Hume
Software Manager – Pickering Interfaces

An ideal test system can be thought of as the sum of its parts—measurement and stimulus hardware, signal switching, cabling and possibly a mass interconnect system, UUT Power supply, External PC or embedded controller, Operating System (OS), and the programming environment. Each part is selected based on parameters such as UUT test parameters, physical dimensions, test times, and budgets.

But one element is missing from the above list – the hardware device drivers. Simply put, the drivers are the last layer between your programming environment and the test systems hardware. In the case of PXI, a VISA driver is required under the standard, so every PXI module will have one. So you should not have to choose, right? Well, Yes & No.

The IVI Foundation (www.ivifoundation.org) has defined a more intelligent device driver standard that many PXI instrumentation and switching companies also support in addition to the mandatory VISA layer. There are advantages for many applications.

So how to choose? In the following pages, we will try and define what a driver is and any OS limitations. Then we'll examine VISA versus an IVI Driver. You will note there is more information on IVI as compared to VISA. As the IVI concept is more complex, it was felt that this would be important to the reader. Also, IVI Switching Drivers will be used as a programming example. The goal is to help you understand the advantages in certain applications so you can determine if IVI will improve your test strategy. Remember that not every PXI Module has an IVI driver. But after reading this article, you will be able to make an intelligent choice of vendor, module, and driver, based on your application.

Operating Systems Supported

The PXI standard requires that PXI modules must support 32-bit Windows® or 64-bit Windows® operating systems; commonly both are supported.

It can be assumed that all versions of Windows supported by Microsoft® will be supported by the PXI vendors, although there may be a lag between the release of a new Windows version and the availability of drivers. At the time of writing most vendors will provide driver support for Windows XP as well as Windows 7 and 8

Support for earlier versions of Windows may also be available, but since these are no longer fully maintained and supported by Microsoft, it cannot be assumed they will be provided in the long term. Note that support for Windows XP ended April 2014 and Windows Vista ended in 2011.

As operating systems evolve, some compatibility problems may occur. For example, Windows 8 requires signed drivers, whereas Windows XP did not, so a driver developed for Windows XP may not install on Windows 8. Always check with the hardware vendor that the operating system to be used is fully supported.

Also take into account that most 32-bit drivers will work on a 64-bit system, so the use of 64-bit Windows does not necessarily dictate the use of 64-bit drivers.

Other Operating Systems

Other OS' may be supported but this is not a requirement of the PXI standard. If the user is planning to use any other OS, checks must be made with the hardware vendors for availability of software support. To successfully operate a PXI platform the operating system must be able to connect to the PXI bus and driver software must be available to support that operating system.

Linux

Linux is increasingly being adopted. However, unlike Windows it is not possible to provide a single driver that will work on any system. The driver must be specifically compiled for the Linux kernel in use. Check with the PXI card vendor for support on the particular Linux system being used. In general, the vendor will need to know the precise Linux distribution being used. Some versions of Linux, particularly real-time versions, are not generally available and may present problems to the PXI card vendor.

Register Level Interface

Where no driver is available for the operating system chosen for the test system, it may be possible to control a PXI card using low-level register level control. This approach requires that the programmer has detailed information of the hardware and control techniques, therefore can only be considered if the PXI module vendor is willing to provide this level of detail.

This route to module control is not recommended except in exceptional circumstances. It is likely to require a great deal of assistance from the module vendor. Check with the vendor before embarking on this approach.

DRIVER MODEL

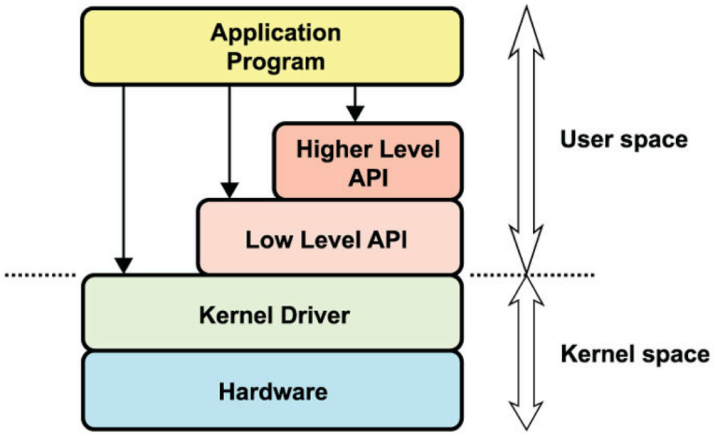
On most operating systems, including Windows, the user cannot interact directly with the hardware but must access through a driver designed for the purpose.

The kernel driver provides the low-level hardware access in kernel space and exposes an interface in user space. The kernel driver provides only a very basic low-level interface and typically a further module Application Programming Interface (API) builds on the kernel to provide an interface better adapted to the control of that particular module.

More advanced APIs may build on the lower levels to provide increasingly useful interfaces to include further features and enhancements.

An application program may access the hardware module using any of the available APIs; choice will depend on a number of factors such as the programming environment, interchangeability requirements, and even personal choice.

This diagram shows a typical set of choices available, from low-level programming using the kernel driver interface through increasing higher-level APIs that provide progressively better modeling of the functionality of the particular hardware module.



A standard Instrument Driver and Programming Layers

VISA is a kernel driver providing hardware control plus resource management. The interface is low level, providing only basic input/output functionality for module control. Module control at this level may be very complex and require detailed understanding of the card hardware. Almost all manufacturers will provide a low-level API that encapsulates specialist knowledge of the hardware module to simplify the programming task.

Many manufacturers provide an IVI driver. This is a higher-level API that builds on the lower driver and may conform to industry standard functionality for the module type.

There may also be more layers than shown above.

In many cases a non-VISA set of drivers will be available. This is useful in cases where VISA is not available, either due to operating system or licensing limitations. For example, VISA is only available on a limited number of Linux distributions so the user will be forced to use an alternate kernel interface.

CHOICE OF DRIVER

A VISA interface driver is required by the PXI standard, however, many PXI modules are provided with a selection of drivers. The user must select the driver most suited to their application and that may also involve an element of personal choice.

Increasingly IVI (Interchangeable Virtual Instrument) drivers are provided. This driver standard is aimed specifically at interchangeability, which is discussed in a later section. It may also be required for particular software tools (notably Switch Executive from National Instruments) that will only handle modules with an IVI Switch class driver.

In some cases a user may elect to construct a system without using VISA. In this case it is essential to consult the hardware vendors to verify if a suitable driver is available.

VISA

The VISA (Virtual Instrument Software Architecture) standard was originally created by the VXI plug & play system alliance and is now maintained by the IVI Foundation (www.ivifoundation.org). The objective of the standard is to define a way of creating instrument drivers with a degree of interoperability between different manufacturers' modules.

The PXI standard encourages the use of the VISA standard; key aspects of VISA are:

- Allows the installation of different drivers from different manufacturers on the same PXI system without conflicts.
- Uses a standardized VISA I/O layer for all I/O functions to ensure interoperability.
- Defines a way of writing drivers.
- A driver that follows the VISA specification uses defined data types and in some cases defined function names.
- Reduces the process of learning new instruments and the time to develop a test system.

IVI

The IVI (Interchangeable Virtual Instrument) standard is supported by the IVI Foundation (www.ivifoundation.org). The aim of IVI is to give a degree of interchangeability, instrument simulation and, in some cases, higher performance. IVI supports all major platforms including PXI, AXIe and GPIB. IVI is a higher-level interface that often uses a lower level driver for the hardware interface; its use may result in slightly slower speed compared to other drivers.

Goals

The stated objectives of the IVI Foundation are to improve hardware interchangeability by:

- Simplifying the task of replacing an instrument with a similar instrument
- Preserve application software if instruments become obsolete
- Simplify code re-use from design validation to production

Improve quality by:

- Establishing guidelines for driver testing and verification

Improve interoperability by:

- Providing an architectural framework that allows users to easily integrate software from multiple vendors
- Providing a standard access to driver capabilities such as range checking and state caching
- Simulating instruments to allow software development when hardware is not available

- Providing consistent instrument control in popular programming environments

As with VISA, IVI is a way to standardize driver development but it goes much further. The set of IVI specifications provides a number of instrument class definitions; each class has a standard interface for programming, including function names and data types.

By appropriate use of IVI class drivers a user can develop a system that is hardware independent, meaning instruments may be easily changed for similar instruments from different vendors without the need to re-code the users application.

At the time of writing the following classes are defined:

IVI-4.1: viScope Class Specification
This specification defines the IVI class for oscilloscopes.

IVI-4.2: IviDmm Class Specification
This specification defines the IVI class for digital multimeters.

IVI-4.3: IviFgen Class Specification
This specification defines the IVI class for function generators.

IVI-4.4: IviDCPwr Class Specification
This specification defines the IVI class for DC power supplies.

IVI-4.5: IviACPwr Class Specification
This specification defines the IVI class for AC power sources.

IVI-4.6: IviSwtch Class Specification
This specification defines the IVI class for switches.

IVI-4.7: IviPwrMeter Class Specification
This specification defines the IVI class for RF power meters.

IVI-4.8: IviSpecAn Class Specification
This specification defines the IVI class for spectrum analyzers.

IVI-4.10: IviRFSigGen Class Specification
This specification defines the IVI class for RF signal generators.

IVI-4.12: IviCounter Class Specification
This specification defines the IVI class for counter timers.

IVI-4.13: IviDownconverter Class Specification
This specification defines the IVI class for frequency downconverters.

IVI-4.14: IviUpconverter Class Specification
This specification defines the IVI class for frequency upconverters.

IVI-4.15: IviDigitizer Class Specification
This specification defines the IVI class for frequency digitizers.

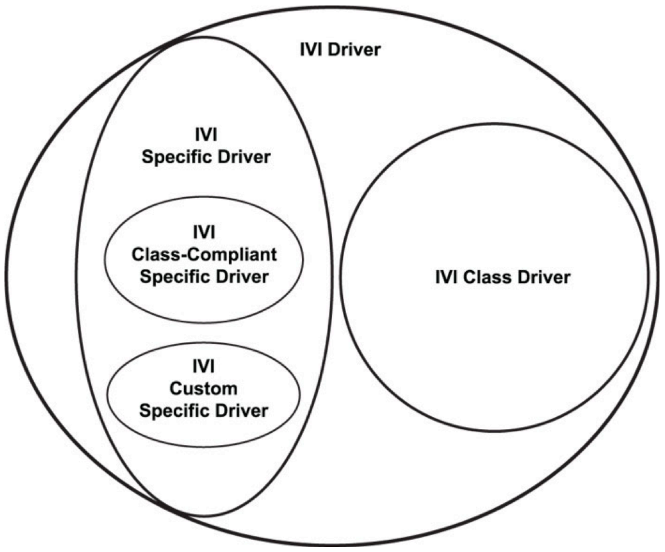
It is important to remember that the class definition cannot include any vendor-specific features; it contains only the basic functionality of the instrument type. It also cannot take into account differences in performance, such as accuracy or speed. In practice it is essential that consideration be given to the consequences of changing from one manufacturer's module to another since those modules may not behave in exactly the same way.

IVI drivers have built-in simulation capability. With this simulation feature is it possible to develop an application without the instrument being present. This means software development may start before instruments are delivered, or while being used in another application.

IVI Driver Architecture

An IVI Driver is a driver that implements the inherent capabilities defined in the IVI-3.2 Inherent Capabilities Specification document, regardless of whether it complies with a class specification.

An IVI Class Driver is a generic abstract class defining the basic features of instruments of that class as agreed by the IVI Foundation members. An IVI Specific Driver contains features specific to a particular vendor that may not be applicable to modules from other vendors. IVI Specific Drivers may be further sub-defined as an IVI Class-Compliant Specific Driver or as an IVI Custom Specific Driver. An IVI Class-Compliant Specific Driver provides both the class functionality and additional vendor-specific functionality.


IVI Driver
IVI Specific Driver
IVI Class-Compliant Specific Driver
IVI Custom Specific Driver
IVI Class Driver

The IVI Driver
(Reproduced from the IVI-3.1 Specification)

Most specifications contain optional class extension capabilities, such as the Scanner function group in IviSwtch. Being optional, it cannot be assumed that all vendors will provide these capabilities.

Most IVI drivers fall into the IVI Class-Compliant Specific Driver group. This means that the driver is class compliant but adds further functionality beyond the class definition.

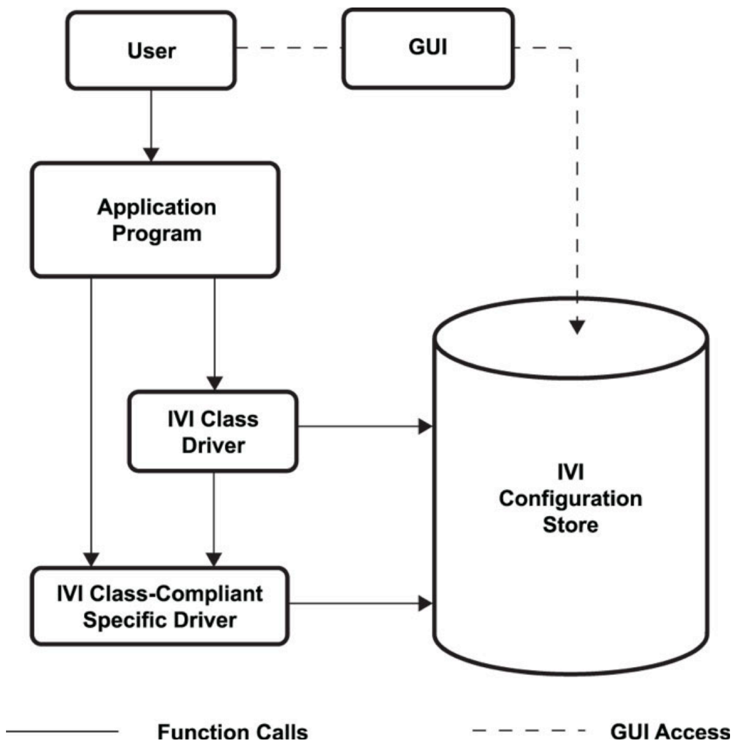
IVI Class Driver	IVI Class Compliant Specific Driver	IVI Custom Specific Driver
Inherent Capabilities	*Inherent Capabilities*	*Inherent Capabilities*
Base Class Capabilities	*Base Class Capabilities*	
Class Extension Capabilities	*Class Extension Capabilities*	
	Instrument Specific Capabilities Functions defined by the manufacturer	*Instrument Specific Capabilities Functions defined by the manufacturer*

Further to the above, drivers may be provided with a C interface, COM interface or .NET interface.

Most development environments are capable of interfacing to a C interface driver, and many to a COM interface; whereas the .NET interface has a more restricted range of environments.

The IVI Configuration Store

Central to the IVI Driver model is the IVI Configuration Store. This diagram shows the relationship between the various software elements involved when using the IVI system.



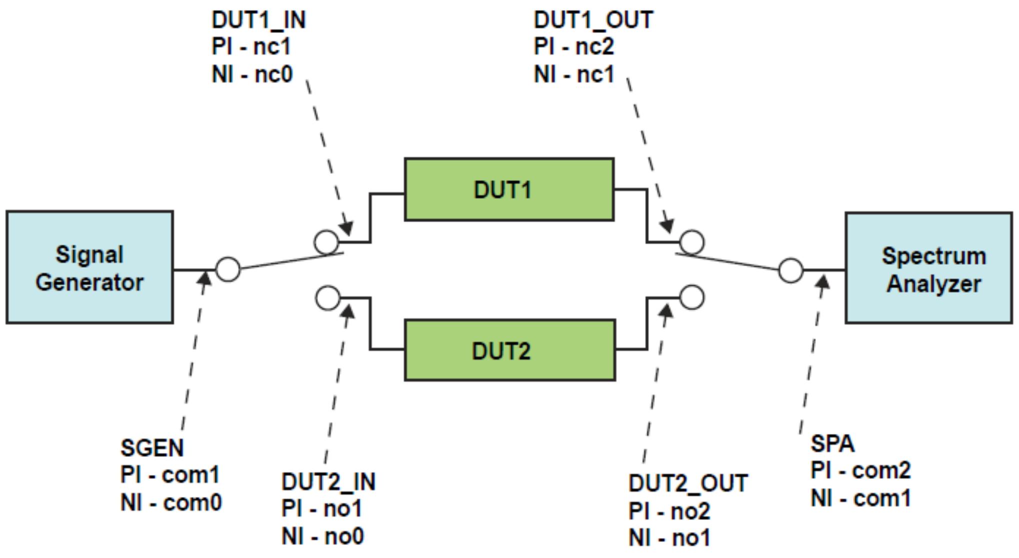
The IVI Driver
(Reproduced from the IVI-3.1 Specification)

The IVI Configuration Store is an XML file containing definitions and relationships between the various aspects of a module and its software driver. The IVI software system provides means to access the store from a driver. Tools to access and manipulate the IVI Configuration Store are available, notably National Instruments Measurement and Automation Explorer (MAX).

EXAMPLE OF INTERCHANGEABLE SWITCH MODULES

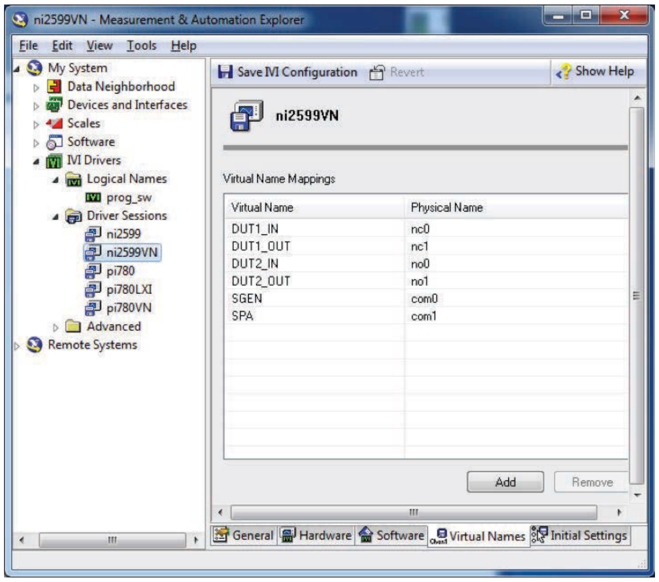
The IVI Switch Class driver is the key to interchangeability. Using this driver allows differences between software implementations from different vendors to be moved from the user application and dealt with by the IVI software system.

In the example shown in the figure, a pair of changeover relays is used to connect one of two devices under test (DUT) to a signal generator and a spectrum analyzer. For this application a coaxial RF switch is required, National Instruments' PXI-2599 and Pickering Interfaces' 40-780-022 are both suitable for this application, however they use different drivers and have different naming conventions for the switch channel names as shown in the diagram.

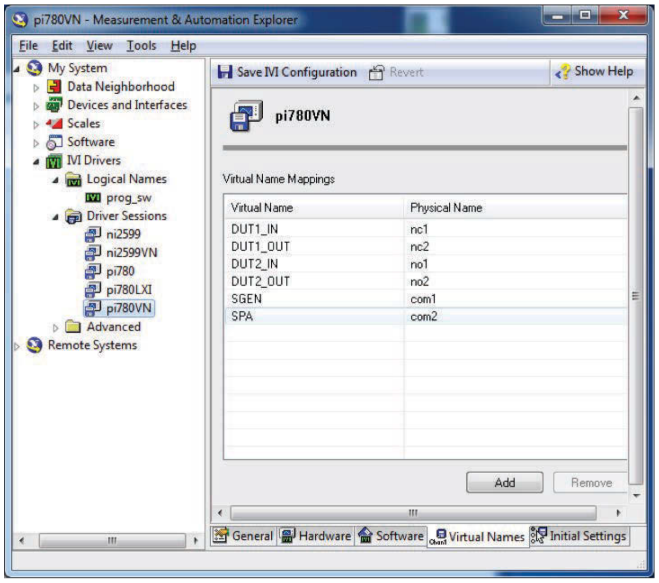


Changeover relays connecting devices under test

The first step toward interchangeability is to define virtual names for the channel names; these virtual names will be employed in the user application. The figure below provides screen shots from NI MAX showing the Virtual Name tables for the NI-2599 and the Pickering 40-780-022 where the differing naming conventions of the two cards are mapped to a common set of Virtual Names.



Virtual Name	Physical Name
DUT1_IN	nc0
DUT1_OUT	nc1
DUT2_IN	no0
DUT2_OUT	no1
SGEN	com0
SPA	com1

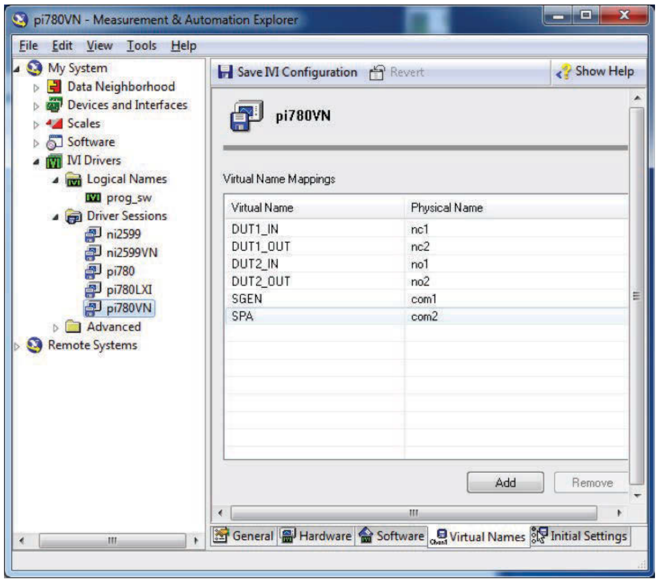


Virtual Name	Physical Name
DUT1_IN	nc1
DUT1_OUT	nc2
DUT2_IN	no1
DUT2_OUT	no2
SGEN	com1
SPA	com2

Defining Virtual Names

Next a level of indirection is employed to decouple the specific drivers from National Instruments and Pickering Interfaces from the user application. The IVI Configuration Store provides this indirection; it creates the concept of a Logical Name which 'points' to a Driver Session, this linkage may be changed within the store such that a Logical Name may be altered to refer to a different Driver Session. So, if all the differences between the NI and Pickering drivers can be encapsulated in a pair of Driver Sessions, then the Logical Name can be simply modified to refer to either Driver Session. The user then creates an application using the Logical Name. If at some time the alternate module is to be used, then the Logical Name may be changed to refer to the alternate Driver Session. So, the switch module may be replaced with one from a different vendor just by changing the linkage of the Logical Name.

In the screen-shot from NI MAX shown below, the logical name ‘prog_sw’ is linked to the driver session for the Pickering 40-780-022.



Virtual Name	Physical Name
DUT1_IN	nc1
DUT1_OUT	nc2
DUT2_IN	no1
DUT2_OUT	no2
SGEN	com1
SPA	com2

A screen-shot from NI MAX

It must be remembered at all times that only differences in software implementations can be interchanged; hardware and performance differences cannot be incorporated.

The user application should code using the IVI Swtch Class Driver thus:

err = IviSwtch_init(“prog_sw”, 0, 0, &vi);

err = IviSwtch_Connect(vi, “DUT1_IN”, “SGEN”);

err = IviSwtch_Connect(vi, “DUT1_OUT”, “SPA”);

This code uses a Logical Name to identify the hardware/software combination and uses Virtual Names to identify the switch terminal channels. It provides completely interchangeable code in that the Logical Name and the Virtual Names may be manipulated in the IVI Configuration Store at any time to permit this code segment to operate different switch modules from different manufacturers without the need to modify the code.

If at some time in the future a new switch product from a different vendor becomes available, all that is required is to create a new Driver Session that defines the driver to be used and the Virtual Name table to define the relationship to the channel names exported by that new driver. The Logical Name may then be modified to link to the new Driver Session and the user application will then use the new module without the need to modify or re-build the application.

Conclusion

As you can see, there are many factors to consider. But it is better to do your research up front to simplify your integration projects. If you need further information, please contact the PXISA and the PXI Vendors being used in your system.

About Alan Hume:

Alan Hume is the Software Manager for Pickering Interfaces, a market innovator in signal switching and conditioning for a broad range of applications and industries.

He graduated from Middlesex Polytechnic (Middlesex University), UK in 1975 with an Honors Degree in Engineering and obtained a Master's Degree in CADCAM in 1990.

Alan has previously held senior design engineering positions with: Marconi Instruments (now part of Aeroflex), Hewlett Packard (now Keysight), Seagate Software and Dage Precision Industries.