

Figure 1 Software Components

## Technical Note

### Software Overview

#### Introduction

The terminology and acronyms within the test and measurement software community can be overwhelming at first glance. For example, how do *VXIplug&play*, *SCPI*, *CVI*, *LabVIEW*, *HP VEE*, *VISA*, *GUI* and *IVI* relate to one another? What does an instrument driver do? How does it differ from a *VXIplug&play* driver? Does it matter anymore whether a VXIbus device is register or message-based? This technical note is intended to provide answers to these questions and more as they relate to communicating with VXIbus instruments.

VXIbus software components can be considered analogous to the software components associated with a desktop computer communicating with an external device such as a laser printer. (Figure. 1)

#### Device to Control

In our analogy, our desktop office device to control is a laser printer, and our VXIbus instrument is a VM2616 waveform digitizer. A user isn't typically given access to the low-level commands for the printer. They are provided with a software driver (graphical user interface or GUI) with their application, MSWord in this example, which provides control over the primary functions (e.g. paper orientation or color). The GUI interfaces with the printer specific hardware driver supplied by the manufacturer (Fig. 2). In the VXIbus instrumentation world, manufacturers will supply a standalone GUI application referred to as soft front panel (Fig. 3). The soft front panel is an application that is developed by a manufacturer that interfaces to their software driver to give very high-level and easy access to instrument functions by imitating the look and feel of a box instrument.

In our desktop office example, the hardware driver would be the basic input/output system (BIOS) on the computer that isolates the low level hardware control of the USB port from the software running on the computer. Each computer manufacturer would have a unique BIOS for that motherboard, and this BIOS would provide a standard software interface to the operating system on board the computer and translate these standard commands to the unique low-level control of the USB port. In the instrumentation environment it has been necessary for each manufacturer of the hardware interface to have their own proprietary

hardware driver. Examples of this: Agilent-GPIB for an Agilent GPIB interface card, or NI-VXI for National Instruments MXI interfaces and embedded controllers. With this approach instrument manufacturers had to develop several instrument drivers for the same device, depending on operating system and hardware interface used.

This becomes cumbersome to support because, unlike the desktop printer user, the instrumentation programmer is often developing code for an automated test environment and will want access to the instrument driver supplied by the manufacturer.

#### VXIplug&play Drivers

The VXI community has worked hard at developing a set of system-level standards for instrument drivers that are classified as *VXIplug&play*. *VXIplug&play* drivers offer interoperability of multivendor software and hardware components and increase productivity by delivering an easy-to-use interface that make the low level I/O communications and hardware drivers transparent to the programmer.

The foundation of a *VXIplug&play* driver is VISA (Virtual Instrument Software Architecture). VISA defines a protocol for I/O communication to VXIbus devices that is supported by multiple vendors. VISA supports non-VXI instrumentation platforms as well and while it is a defined set of standards, the version of VISA will be dependent on the instrument control interface. For example, if there is an Agilent IEEE488 controller card interfacing to a Tektronix scope, Agilent-VISA would need to be installed on the PC. Conversely, if a National Instruments VXI-MXI2 card is interfacing to a VXI

## Technical Note



Printer Driver GUI  
Figure 2

Technology VM2616 digitizer, NI-VISA would be installed. Both manufacturers version of VISA utilize the defined set of standards which brings true interoperability. In other words, a *VXIplug&play* driver will work with either an Agilent or NI VISA layer. VXI Technology is committed to supplying products that will work with Agilent or NI's VISA.

### Message vs. Register-based Commands

When a *VXIplug&play* driver does not provide the flexibility required by a 'power-user', it might be necessary to communicate to the device through the instrument's lower-level interface. The reasons for this are:

- **Complexity of instruments:** For complex instruments, it is difficult to develop a driver for every permutation.
- **Test program speed:** An instrument driver will always slow down test throughput.
- **The need for flexibility:** Instruments are used in many different applications, and it may become necessary to access them in a particular manner as opposed to be limited to the driver.

Low-level communication may be in the form of ASCII strings for 'message-based' devices, or through direct access to device registers for devices that are 'register-based'.

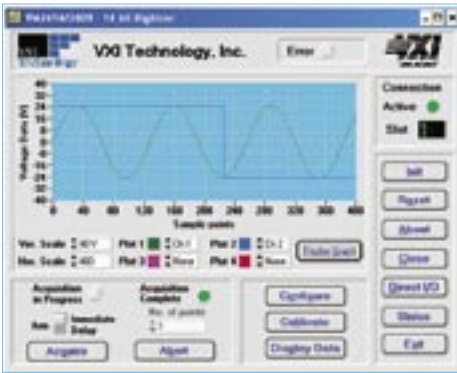
Message-based commands are also referred to as 'word serial' and form another layer between the user and the hardware as messages must be parsed via on-board local intelligence. SCPI (Standard Commands for Programmable Instruments) is a derivation of the word serial protocol and was introduced to provide homogenous control over instruments eliminating the need to learn proprietary commands for each device. Register-based devices remove all layers and users setup measurement parameters by providing access to bits in hardware registers. Register-based devices do not support SCPI.

The general rule of thumb has always been that message-based devices are easier to program but register-based devices are most efficient in transferring data to and from device memory. Pseudo-register devices give the user flexibility for optimizing test development and execution time by providing message-based communication for setting up devices and access to registers for retrieving data. This also highlights one advantage in programming through the plug&play driver interface because the message and register command layer become transparent to the user. In fact, SCPI compatibility has become much less important to users of VXI devices with the maturation of plug&play drivers.

### Hardware Interfaces

In our analogy, the hardware interface is the USB2.0 port in our desktop office example, and a MXI-2 interface in the VXIbus example. As with printers, there are many ways to interface a PC with VXIbus devices, but the most common are:

1. IEEE-488.2 (GPIB) to VXIbus. With this approach a GPIB interface card is housed in the external computer, and is connected to a GPIB/VXI Slot 0 in the VXIbus mainframe. This is an 8-bit parallel protocol, therefore, it's typically the slowest way of communicating between a remote host and the VXI mainframe (about 1 MB/s).
2. MXI-2 Interfaces. With this approach a MXI-2 interface card is housed in the external computer, and is connected to a MXI-2 Slot 0 in the VXIbus mainframe. This approach extends the VXIbus backplane to the external computer. It is a 32-bit protocol, and offers some of the fastest transfer rates between a remote PC and the mainframe at around 25 MB/s.
3. IEEE-1394 (FireWire) As with the previous two interfaces, this requires a remote PC. An IEEE-1394 Slot 0 interface card in the VXI mainframe connects to the 1394 port on the external computer. This is a serial protocol that operates at about 400 Mb/s. 15 MB/s Transfer rates between the remote and the mainframe can be achieved, though because of protocol overhead, maximum system speed is seen when transferring large blocks of data as opposed to sending discrete commands.



VXI Instrument GUI  
Figure 3

## Technical Note

### Software Overview

4. Embedded Controllers. This approach embeds the controller within the VXIbus mainframe (typically in Slot 0). Embedded controllers can be Pentium computers, Motorola 68XXX based controllers or other platforms. The hardware interface to the VXIbus backplane is built into the embedded controller with this approach, similar to a parallel port.

It is important to point out that applications written using *VXIplug&play* drivers require negligible modification when switching between hardware interfaces. This is a huge advantage that the VXI platform has over other similar platforms as it is truly independent of computer architectures and can easily adapt to the latest technology such as USB 2.0 and ethernet. Companies that have made significant investment in software development on systems using one type of interface can maintain that investment as newer interfaces are introduced to the market.

### Application Development Environments

The application development environment (ADE) is referred to as the language or environment in which the user would develop the program that controls and coordinates all of the instruments within his system. Examples of application development environments are:

MS C++ and Visual C++, Borland C++, MS VisualBasic, LabWindows/CVI, and LabVIEW and HP VEE.

LabVIEW and HP VEE are referred to as graphical programming languages that do not resemble classic scripted language formats. These two languages, along with LabWindows/CVI have been developed primarily for the test and measurement community, while the other languages are commonly used in a variety of industries. *VXIplug&play* drivers will work with any of the application environments listed above, in addition to many others.

There is another software element that is commonly found in ATE systems which is referred to as the test executive. The test executive manages system level utilities such as test sequencing, data logging, and

printing functions. A test executive will interface to software modules written in one of the ADEs. Examples of test executives are Teradyne's TestStudio and National Instruments TestStand. ADEs and test executives are typically mutually exclusive entities, therefore, there is flexibility in selecting a programming language and test executive to best meet requirements. For example, a system architect may want to develop in Visual C++ because of its power and wide acceptance, while selecting NI TestStand as the test executive.

Each ADE and test executive has its own advantages and disadvantages, which go beyond the scope of this technical note.

### IVI Drivers

The Interchangeable Virtual Instruments standard (IVI) is a relatively new specification that has been introduced primarily as a means of isolating the application code from the specific instrumentation within a system. IVI drivers are divided into various classes based on instrument functionality (e.g. IviScope) and builds on the *VXIplug&play* specification by providing features such as state caching, that improve system performance. IVI drivers are intended to allow an ATE developer to replace one vendors instrument with one from another vendor within the same class without any modification to application code.

An instrument class accounts for the most common features and functionality within that class. For example, the IviScope class defines channel counts, possible trigger sources, and sample record lengths in addition to other common scope functions. Digitizers could also fall under the realm of this class as it is defined. It is important to note that IVI drivers can be provided with either a C or COM application interface which are optimized for specific development environments and it is up to the vendor whether they want to support one or both of the interfaces.

As of this writing, instrument class definitions continue to evolve. Some of the most common classes have been defined. For the latest information regarding IVI, refer to

ANSI-C	MS C++ Borland C++ Lab Windows/ CVI	MS C++ Borland C++ Lab Windows/ CVI	MS C++ Borland C++ Lab Windows/ CVI	Lab Windows/ CVI Sunsoft cc Sunsoft CC	POSIX c89 HP CC
	MS C++ Borland C++ MS VB Lab Windows/ CVI LabVIEW HP VEE	MS C++ Borland C++ MS VB Lab Windows/ CVI LabVIEW HP VEE	MS C++ Borland C++ MS VB Lab Windows/ CVI LabVIEW HP VEE	LabVIEW HP VEE Sunsoft cc Sunsoft CC	LabVIEW HP VEE POSIX c89 HP CC
Run-time Link	LabVIEW	LabVIEW	LabVIEW	LabVIEW	LabVIEW
G	WIN	WIN95	WINNT	SUN	HP-UX

# Technical Note

## Software Overview

[www.ivifoundation.org](http://www.ivifoundation.org).

### Application Programming Examples : Setting the range on a digitizer

Using a VXIplug&play driver function call:

```
vtvm2616_set_voltage_range(instrHandle, vtvm2616_VOLTAGE_RANGE3, numChannels, channelList)
```

where instrHandle is a unique session to a particular VM2616, vtvm2616\_VOLTAGE\_RANGE3 is a constant defined in a header file that equates to the 1 V range, numChannels and channelList are variables defined in the application code defining the channels that will be set to the 1V range.

Using VM2616-specific word serial command through the VISA layer

```
viWrite(instrHandle, "volt:rang 1, (@1:4)", count, &returnCount)
```

In this example, "volt:range 1, (@1:4" is a VM2616 specific word serial command in ASCII format which sets the range for channels 1-4 to 1 V, count and returnCount are variables associated with the size, in bytes, of the ASCII string.

Message-based devices utilize word serial commands. To program a register-based digitizer making calls directly to the hardware register, it is necessary to have a register map. Writing to a register is typically done in 16 bit blocks, and often multiple functions are contained within a single 16-bit register. For example,

the SVM2608 is a register-based digitizer.

The register that contains the bits that set the range also contains bits that set the low-pass filter and trigger source. In this particular example, bits D5-D7 must be set to 0x3 to achieve the 1 V range. To ensure that the other functions in this register are not affected by writing to D5-D7, the register must first be read, and then masked such that only D5-D7 are modified.

Using SVM2608-specific register-based commands through the VISA layer

```
viOut16(instrHandle, VI_A24_SPACE, 0x80, 0x30)
```

Here, VI\_A24\_SPACE is a constant defining the base address of the memory allocated to the instrument, 0x80 is the offset of the target register, and 0x30 is the value to be written corresponding to a 1V range. Note that using a pnp driver would remove the need to dig into register maps at this level, greatly simplifying the programming task at hand, though test execution times may suffer.

Using the IviScope class driver

```
IviScope_ConfigureChannel(instrHandle, channel, range, offset, coupling, probeAttenuation, enabled)
```

where instrHandle again is the communications portal to the generic digitizer module, and the rest of the parameters are commonly configured attributes of oscilloscopes/digitizers. The difference here is that there is nothing specific at this layer that ties the application to a particular vendor's instrument.