A Comparison of Leading Switch/Measure Solutions

Application Note

This application note compares the features, execution speed and ease of software development for switch/measure solutions used in functional test and data acquisition environments. It examines the Keithley 27xx, Racal 1256, Agilent 34970A and Agilent 34980A Switch/Measure Units, as well as an Agilent 3499A/34401A combination, Agilent E1411B/ E1476A VXI combination, and a National Instruments PXI-4070/SCXI-1128 PXI combination in the Visual Studio.NET application development environment.

Switch/measure system characteristics

While a number of complex measurement functions can be present in a test system, two components are almost always present – a digital multimeter (DMM) and a bank of relays (switches). A fundamental core, then, is a switch/measure function. This can be implemented in three different ways:

- Discrete instruments with cable interconnects, such as an Agilent 34401A standalone DMM or E1411B VXI DMM connected to an Agilent 3499A/B/C switchbox or Racal 1256 switchbox.
- VXI or PXI or PXI-hybrid mainframes, such as the Agilent E1411A VXI DMM and an E1476A VXI switch card, or the NI PXI-4070 DMM and SCXI-1128 switch card (the latter being a hybrid configuration in which control signals are shared between the PXI backplane and the SCXI backplane, either in a combo chassis or through a cabled arrangement from a PXI chassis to a standalone SCXI chassis).
- Dedicated instruments with proprietary backplanes containing a DMM and a variety of switch cards, such as the Agilent 34980A, Agilent 34970A, and Keithley 2701 Switch/Measure Units.

Each type of solution has pluses and minuses, which will be discussed in depth here. However, the use model of this equipment generally falls into two categories: Data Acquisition and Electronic Functional Test. Which solution you choose will depend greatly on your intended use.

The difference between data acquisition and Electronic functional test

The same voltmeter and switch can be used in two types of test systems:

- **Data acquisition (DAQ),** in which numerous readings are taken to characterize performance of a mechanical, electrical or electronic device, such as measuring the temperature at thousands of points on a spacecraft as it re-enters the atmosphere
- **Electronic functional test (EFT),** in which stimuli are applied to an electronic module called a Device Under Test (DUT) and the outputs monitored for expected responses and compared to a set of limits, such as determining whether an engine control module for an automobile is operating correctly

The throughput that is achievable with the same hardware differs greatly between these two environments. In DAQ mode, a voltmeter and a bank of relays are programmed together to do a scanned measurement – a list of switch open/close states is downloaded into the switchbox, and a hardware handshake links the DMM measurement to the appropriate switch setup. A measurement complete trigger from the voltmeter can be used to make the switches advance through this list or they can simply free run. The DMM can either wait a preprogrammed delay after it sends out a measurement complete trigger or it can wait until it receives a hardware signal from the switches called an advance trigger, indicating that the switches have gone to their programmed state and it is OK to take a reading. Because of this hardware handshaking, it is possible to minimize the amount of I/O being transferred to and from each instrument. In addition, the setup time is not duplicated for each measurement; setup information is downloaded and then a single initiation command starts the whole operation. The execution speed can thus be very high—on the order of 1,000 readings/sec depending upon the desired measurement resolution and the switching speed. In such systems, high-speed backplanes can improve throughput, since a lot of data has to be transferred. VXI and PXI systems shine here.

In an EFT system, the situation is different. Stimuli must be applied and single measurements taken, and this process is repeated many times to cover all pins on the DUT. Thus the overhead of individual readings and reconfiguration of the stimulus and measurement instruments must be incurred repeatedly (close a relay, take a reading, open a relay). Polling is often done to make sure a reading is ready, which can add extra I/O execution time. The result is that execution speeds are more like 100-500 readings per second. Since fewer readings are made, a high-speed backplane does not help much here, which makes the lower cost dedicated switch/measure solutions a better choice.

Cardcages

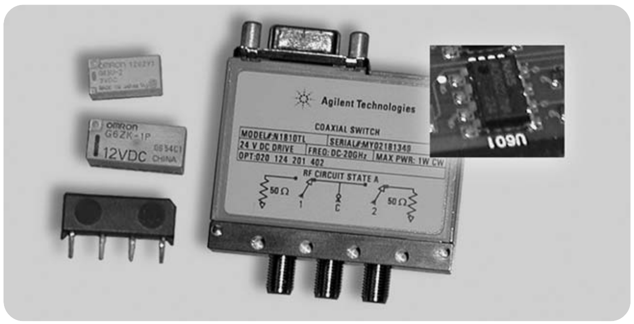
All switching systems on the market are implemented as cardcages, with a variety of card types offered. Some have open architectures, meaning that the backplane interface is documented and implemented by more than one vendor, such as VXI and PXI. Others have vendor-specific internal backplanes, such as NI's SCXI, Racal's 1256, Keithley's 27XX and Agilent's 3499 family, 34970A and the new 34980A. The reason many vendors prefer their own designs is so that they (and you) do not have to pay for the extra capabilities demanded by open standards. They can put just enough power, just enough cooling, and just enough backplane speed in the box to optimize it for the switch cards that they intend to offer. Vendor-specific architectures often also allow others to provide customization by means of a breadboard card that typically has address decoders already implemented, with room for users to add their own circuitry.

In addition to relays and DMMs, there are several other classes of functionality typically implemented in switch/measure systems because they are often used together. These are:

- Digital I/O (DIO)
- Digital to Analog converters (DACs), which are sometimes fast enough to use as waveform generators at low frequencies
- Frequency counters/totalizers
- Isothermal Terminal Blocks
- Customizable (breadboard) cards (useful for signal conditioning), such as that typically required by strain gages

The switch cards come in a variety of types, and are configured as multiplexers, matrices or general purpose. These are:

- Low bandwidth (DC-100 MHz)
  - FET relays (high switching speed, high on-resistance, low voltage and current)
  - Reed relays (medium switching speed, low on-resistance, medium voltage and current; also available with low thermal EMF)
  - Armature relays (low switching speed, low on-resistance, high voltage and current; also available with low thermal EMF)
- High bandwidth (up to 18 GHz)
  - RF relays (low speed, low on-resistance, low voltage and current)
  - Microwave relays (low speed, low on-resistance, low voltage and current)



Ease of programming

Application development environments

In any test system, it is necessary to program the instruments. Several Application Development Environments (ADEs) are in common use:

- Graphical Languages:
  - Agilent Technologies VEE Pro
  - National Instruments LabVIEW
- Textual Languages
  - Microsoft Visual Studio.NET (VB/VC++ 7.0, C# and many more)
  - National Instruments LabWindows/CVI
  - Previous versions of Microsoft languages (VB/VC++ 6.0, etc.)
  - Rocky Mountain Basic ("HP Basic")

Of all of these, the three most commonly used are VEE Pro, LabVIEW and Visual Studio 6.0, with Visual Studio.NET making gains. In the Visual Studio.NET environment, Agilent offers a software product called T&M Programmers' Toolkit and NI offers a product called Measurement Studio, both of which make using instruments easier. In the benchmarks performed for this application note, all instruments were programmed using Visual Studio.NET using both Agilent T&M Toolkit and NI Measurement Studio.

Drivers vs. SCPI

Standalone instruments usually require ASCII commands to be sent to them over an interface such as GPIB, USB or LAN. The command language has been standardized and is called SCPI (Standard Commands for Programmable Instruments). These commands can be sent to the instrument either directly via low-level function calls that talk to the appropriate interface, or through higher-level function calls. These higher-level calls are contained in software called Drivers. You can write these yourself or you can use drivers provided by the manufacturer or others.

Cardcage-based systems such as VXI and PXI are most often controlled via data registers rather than ASCII commands (although there are a few message-based cards for VXI that rely on the built-in word-serial protocol that is a part of the VXI specification). Although one could send the necessary data to these registers using the same low-level interface function calls mentioned above, it would take quite a while to write such a program, so it is more customary to use a driver provided by the manufacturer. It is also possible to send SCPI commands to some register-based instruments – Agilent provides Downloadable SCPI Drivers (D-SCPI) for some of its VXI cards. This code resides in Flash ROM on a GPIB controller (E1406A Command Module) located in slot 0 of the VXI mainframe. Agilent also offers Interpreted SCPI (I-SCPI) for its VXI FireWire interface. I-SCPI is typically much faster than D-SCPI since it uses the faster command parsing of the PC. PXI does not provide for a similar arrangement; all PXI cards are register-based and thus require drivers to be provided by the manufacturer. Drivers for VXI and PXI instruments are provided as a compiled DLL for use with any language, and may also be offered as special drivers for Agilent VEE Pro or NI LabVIEW.

Except for PXI cards, which require drivers, most drivers do not include all possible instrument functionality. There is often a "pass-through" function available though, allowing SCPI commands that are not implemented in the driver to be sent to the instrument from the driver.

In the Visual Studio environment, Microsoft's IntelliSense feature makes using a driver quite easy, since the programmer simply selects easy-to-grasp names from a drop-down list and views a description of the function and its parameters.

The primary advantages to drivers, at least where there's a choice of SCPI or drivers, are:

1. The program is more transportable and more readable by other programmers. While the SCPI syntax is widely understood, figuring out exactly what string to compose almost always requires a visit to the instrument's manual. It is also worth noting that a driver will eventually generate SCPI commands for you, so you may need to learn it anyway as you debug your application.

2. Built-in help during development via IntelliSense (VB6 and .NET)
3. Built-in state caching, if implemented, can improve execution speed (discussed in more detail in the next section)
4. Quirkiness of the instrument may have been taken into account for you, reducing the chance that you will need support.

The primary disadvantage to drivers is that, except for PXI as mentioned above, they may not include all possible functionality. Another is that if the driver does not function correctly, it can be difficult to debug. The task is made simpler by using the I/O Monitor available in Agilent T&M Toolkit to observe the actual SCPI commands being sent to the instrument. However, that means that you must understand SCPI in the first place, and you may have chosen to use a driver so you didn't have to do that. This is an argument for using drivers that are released and supported by the instrument manufacturer rather than those of independent 3rd parties.

Interface drivers

Before your program can pass commands to a device, a driver must be written to control the hardware interface, such as LAN, USB, GPIB, MXI-3, and FireWire. Low-level drivers to handle such hardware are shipped as part of the vendor's I/O libraries. NI calls their interface drivers Passport drivers, while Agilent calls theirs Tulip drivers. We will not go into detail on these here because you will not have any reason to use them directly. However it is useful to know they're there, especially when using NI and Agilent hardware and software in the same environment. It is possible for NI software to access Agilent interface hardware and Agilent software to access NI interface hardware. This is described in detail in the Agilent I/O Library online documentation.

VISA, VXIplug&play and IVI

Software drivers are organized into layers. The simplest standardized layer is called VISA (Virtual Instrument Software Architecture). (It could be argued that VISA is not really a driver since it is so generic, but let's not split hairs). It allows SCPI commands to be sent to the instruments via simple C function calls such as viRead and viWrite, and binary commands to be sent to registers via C function calls such as viPeek and viPoke. Agilent and NI are the two major providers of this software. Agilent also provides a library called SICL (Standard Instrument Control Library), which predates VISA and is in fact used as the underlying code for Agilent's implementation of VISA. Agilent has also adopted the industry standard VISA-COM architecture by providing its VISA C function calls to "COM-friendly" environments such as Visual Studio. [COM is an acronym for Component Object Model, a Microsoft standard for encapsulating libraries, making it easier to interface with and maintain them.]

Register-based devices, including most VXI instruments and all PXI instruments, require drivers unless the developer chooses to use the viPeek and viPoke commands mentioned earlier, which is a very tedious process. A layer on top of VISA was created for this purpose, called VXIplug&play. Although the name has VXI in it, the concept has long been extended to non-VXI instruments, including most standalone programmable instruments.

VXIplug&play drivers are instrument specific – the name of the instrument is contained in the function names. A typical C example is:

```
hp34401_read_Q (vi, readings, numReadings);
```

(Many Agilent drivers were written when Agilent was a part of Hewlett-Packard, and the driver names were not changed to maintain backwards compatibility.)

Since many products have similar functionality, unnecessary duplication of functions resulted from the VXIplug&play standard. To fix this, instruments were lumped into classes, such as DMM, Scope, Function Generator and Switch, and drivers for these classes were named IVI (Interchangeable

Virtual Instruments). Definitions for these drivers were standardized by an organization called the IVI Foundation. There are two forms of IVI:

- IVI-C, preferred by NI. Different versions are required depending upon the language (C, LabVIEW, etc.). A typical C example is:

  ```
  IviDmm_ReadMultipoint (vi, maxTime, arraySize,
  readingArray, &actualPoints);
  ```

- IVI-COM, preferred by Agilent due to its applicability to many development environments without requiring different versions for each environment (all Microsoft COM environments including VEE, LabVIEW, Visual Studio and more). An example in Visual Basic is:

  ```
  myDMM.Measurement.ReadMultiPoint (maxTime, readings)
  ```

In both cases, it is possible to use instrument-specific function calls to handle cases in which the class-specific functions do not map into the available functionality of an instrument. Using such functionality removes the "Interchangeable" from IVI, but sometimes that is why people buy one instrument over another—to get functionality not available elsewhere. You must decide if instrument interchangeability is a requirement and write your code accordingly.

VXIplug&play and IVI drivers for faceless instruments are shipped with code that creates a soft front panel, which is a standalone executable program. It cannot be integrated into an instrument control program that you write. However, some of them can generate code snippets that can then be cut and pasted into certain development environments.

VXIplug&play drivers are also shipped with a function panel, which simplifies the calling of C functions from Agilent VEE Pro and LabVIEW.

LabVIEW and VEE Pro drivers

LabVIEW uses a special type of graphical driver specific to NI, officially called a G Framework VXIplug&play driver. These are also referred to as "G" , "G-Win", LabVIEW or "LabVIEW Certified Plug and Play" drivers. However, since LabVIEW can also use the Windows framework (C-based) VXIplug&play drivers, it is often difficult to tell which type of driver you are downloading until you can open it and look at it.

Agilent's VEE Pro can also use a variety of drivers, including VXIplug&play (Windows framework), IVI-C and IVI-COM drivers. In the past, Agilent developed a special type of driver, called a "Panel" driver, that was specific to VEE Pro. Today, Agilent has standardized on COM and .NET-friendly drivers for all new instruments to provide more development environment choices for customers.

What about speed?

Experiments with the hardware used in this benchmark have shown that VXIplug&play and IVI drivers can be as fast as SCPI (via VISA) in modern computers. Although there are some caveats to this, there is no reason to avoid drivers solely due to concerns about execution speed.

No matter which way you go, there is a more insidious problem with instrument programming, and that is sequencing of commands. If you use SCPI yourself, you need to understand that some commands cause other instrument states to be changed. A driver may take care of this for you, at the expense of having to send a lot of query commands to the instrument to figure out what state it is in or by sending extra commands to the instrument to make sure it is in a given state. This can add a fair amount of execution time. The problem can be solved by the use of state-caching, which is commonly implemented in IVI drivers and which you can implement yourself if you are so inclined. A state-caching driver keeps track of the current state of the instrument, and when a command comes along that would not result in any change in state, it doesn't send that command to the instrument, thus saving command transfer time and command parsing time.

State problems can generally be fixed by issuing a reset command to the instrument so that it reverts to a known state. However, reset commands can take quite a long time to execute within an instrument, so it is good practice to issue only one reset command the first time a program is run and to manage the state carefully as the program unfolds, making sure that the instrument is in a state equivalent to that of a reset command at the end of the program. It is also usual practice to issue a reset command when errors are encountered.

Here's an example of redundant commands that can add to execution time: suppose you program in SCPI and send the command "CONF:VOLT DC 10,.001", which will tell a DMM to go to the DC function and set the range to 10V and the resolution to 1 mV. At top GPIB speeds of 1 MB/sec, this 21-character string (including the carriage return character) would take 21 microseconds plus the overhead involved in the function calls, which would add about 10us in a modern PC. The instrument has to parse this command using its considerably slower processor. In the case of a 34401A DMM, for example, the 12 MHz processor would take about 21 milliseconds of additional time. Now, if the DMM is already in that state, this is 21.031 ms of time that did not need to be spent. It may not sound like much, but a typical test program can have hundreds or thousands of DMM readings, so it is important to maximize the efficiency of the I/O and subsequent processing in the instrument. An IVI driver that has been implemented with state caching would know that this is the state the instrument is in and would not bother to send the command. You can implement state caching yourself at the expense of extra programming time. By the way, this example shows why LAN and USB are not appreciably faster than GPIB for small data transfers. The I/O transfer speed is so much faster in all cases than the instrument processing time that the choice of interface does not yet matter very much.

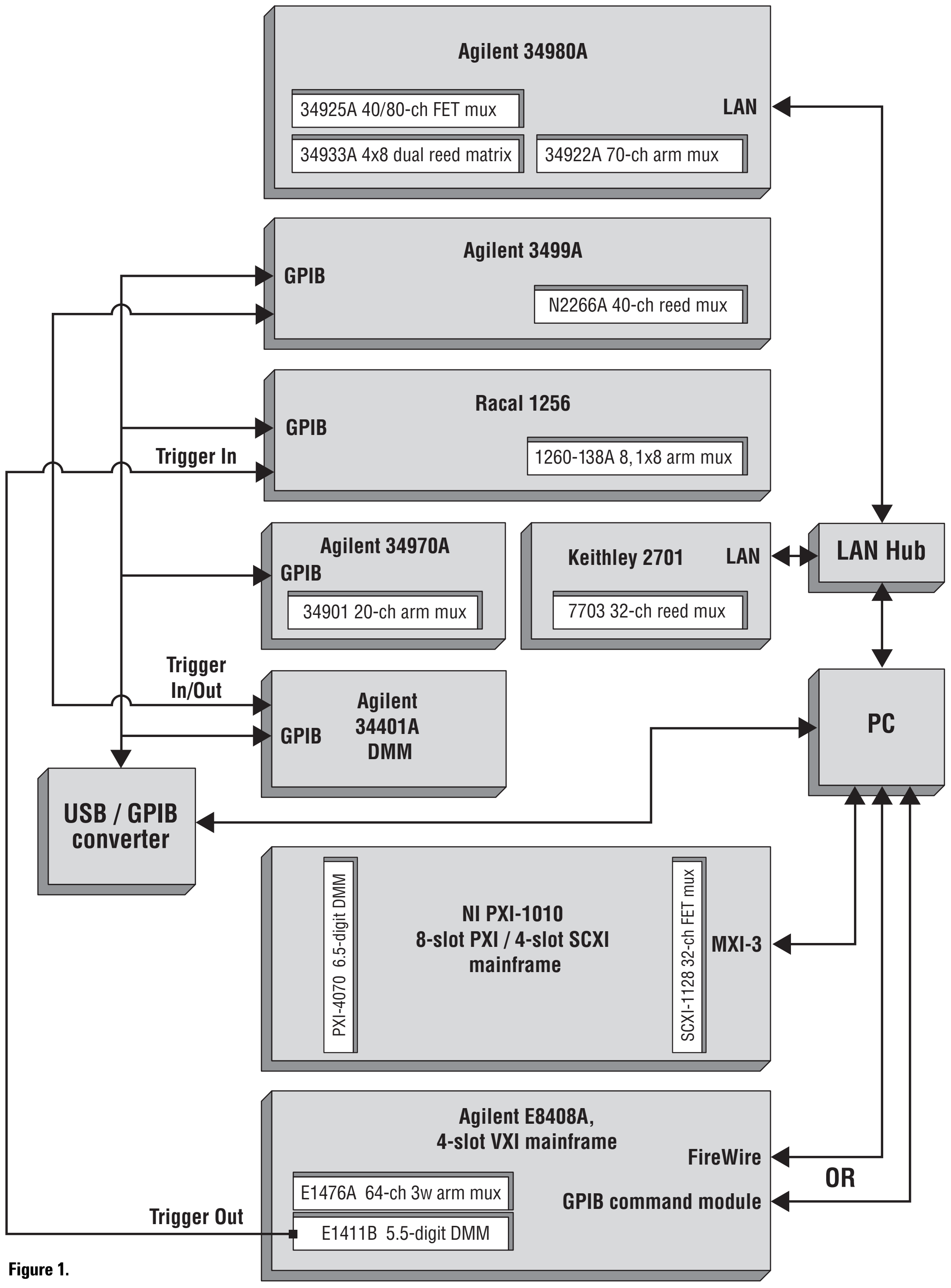
The Agilent 34980A (below) is a new class of instrument that uses a much faster internal microprocessor. It can execute that same string in about 1.5 ms, reducing the need for state caching.

Common Development Environment

In order to compare apples to apples while benchmarking the various instruments discussed in this app note, it was necessary to use a common development environment. Although LabVIEW and VEE Pro are popular graphical environments, much manufacturing test work is done in textual environments. In this realm, VB6 and C or its derivatives (VC++, LabWindows/CVI) dominate. However, there are numerous reasons to develop new applications using Microsoft's Visual Studio.NET environment. Both Agilent and NI have released tools to assist a test system developer create programs quickly in this environment (Agilent's T&M Toolkit, and NI's Measurement Studio). Microsoft will also make it essential for developers to upgrade to .NET as support for the older programs wanes. VB6 programs can also be ported into the VB.NET environment using .NET's built-in porting tool. For these reasons, VB.NET was chosen as the common development environment.

It was also desired to show that NI, Keithley, Racal and Agilent hardware could work together in a real test system. Thus, a test system was constructed using all of these products, as shown in Figure 1 (page 7). LAN was used for some instruments – the Agilent 34980A and the Keithley 2701 via a LAN hub. MXI-3 was used to connect the PC to the PXI cage. FireWire was used to connect the PC to the VXI cage. Since the VXI cage can also be controlled via a GPIB command module, that was also tried, via a dedicated PCI GPIB interface card. Other instruments were controlled via GPIB, using an Agilent 82357A USB/GPIB converter.

Figure 1 also shows the relative height of each of the devices tested, ranging from 2 EIA units (an EIA unit, or "rack unit" is 1.75 inches) for the 34401A DMM to 4 units for a "3U" PXI/SCXI cardcage. (The PXI cards are 3U, but the cardcage they go into is 4U.) No attempt was made to compare VXI rack sizes. A 4-slot version (3U) was used for convenience, although a 13-slot frame (7U) is more typically used in large test systems. It also points out that although new systems today tend to be built using non-VXI hardware, a 4-slot frame is a great solution for the occasional card whose functionality is only available in VXI.



Agilent 34980A
34925A 40/80-ch FET mux
34933A 4x8 dual reed matrix
34922A 70-ch arm mux
LAN
Agilent 3499A
GPIB
N2266A 40-ch reed mux
Racal 1256
GPIB
Trigger In
1260-138A 8, 1x8 arm mux
Agilent 34970A
GPIB
34901 20-ch arm mux
Keithley 2701
LAN
7703 32-ch reed mux
LAN Hub
Trigger In/Out
Agilent 34401A DMM
GPIB
PC
USB / GPIB converter
PXI-4070 6.5-digit DMM
NI PXI-1010
8-slot PXI / 4-slot SCXI mainframe
SCXI-1128 32-ch FET mux
MXI-3
Agilent E8408A,
4-slot VXI mainframe
FireWire
OR
GPIB command module
E1476A 64-ch 3w arm mux
E1411B 5.5-digit DMM
Trigger Out

Figure 1.

Benchmark results

A program was written in VB.NET that controlled all instruments in both a Data Acquisition mode (scanned voltage readings) and EFT mode (single reading with associated switching). The following software was installed on a Compaq Evo with a 2.4GHz Pentium 4 with 512MB of RAM:

- Agilent's T&M Toolkit 1.2
- The above Agilent software installs I/O libraries, including SICL and Agilent VISA, and .NET wrappers for all Agilent instruments that have VXIplug&play drivers, including the E1411B VXI DMM and E1476A VXI switch
- VXIplug&play drivers were downloaded from the Agilent Developer Network (ADN) web site for the E1411B and E1476A.
- D-SCPI drivers were downloaded from the ADN web site for the E1411B DMM and E1476A switch
- IVI-COM drivers were downloaded from the ADN web site for the 34401A DMM, 3499A Switch Unit and 34970A Switch/Measure Unit
- HP VIC (not renamed since HP and Agilent split) was downloaded from the ADN web site in order to initialize a GPIB Command Module for use with D-SCPI calls.
- NI Measurement Studio.NET 7.2
- An IVI-C driver was downloaded from the Keithley web site for the 2701 Switch/Measure Unit
- An IVI-C driver was downloaded from the Racal web site for the 1256 Switch unit
- NI-SWITCH, NI-DAQ, NI-DAQmx and NI-DMM IVI-C drivers were installed from the NI driver install disks and the Toolkit Driver Wrapper Wizard was run to create .NET compatible calls

Benchmark timing results:

EFT time = time to open and close one relay and trigger and read one DMM DCV reading on the 10V range with a resolution of 1mV (4.5 digits). Reported time is average of 20 such measurements

DAQ time = time per reading to scan a 20-channel list, taking a measurement as each relay closes.

	Driver		SCPI	
	EFT	DAQ	EFT	DAQ
Agilent 34980A				
70-ch armature mux (34922A)	16.9 ms	10.4 ms	15.1 ms	10.1 ms
Dual 4x8 reed matrix (34933A)	9.6 ms	*1*	8.4 ms	*1*
40/80-ch FET mux (34925A)	10.0 ms	2.7 ms	7.9 ms	2.7 ms
Keithley 2701-7703				
32-ch diff. reed mux	440 ms	68 ms	n/a	n/a
Racal 1256-138A/E1411B				
8 1x8 2-wire armature mux	4.13 ms	113 ms[2]	n/a	n/a
Agilent 34970A-34901A				
20-ch armature mux	52.2 ms	22.8 ms	70.0 ms	25.2 ms
Agilent 3499A-N2266A/34401A				
40-ch reed mux	29.5 ms	21.4 ms	35.3 ms	27.1 ms
Agilent VXI E1476A/E1411B				
64-ch 3-wire reed mux	30.1 ms	11.9 ms	46.6 ms[3]	2.94 ms[3]
NI SCXI-1128/PXI-4070				
32-ch FET mux	12.8 ms	2.91 ms[4]	N/P	N/P

Notes:

1 *It is not possible to do a "scanned" measurement using a matrix. This is not the typical use model for a matrix, which is used most often for EFT testing.*

2 *The Racal 1256 took 2 seconds to process the "define scan list" command. That is why its data acquisition time was so long. Without that, the execution time would have been about 13 ms. Of that, 10 ms was the "trigger delay" parameter that was used, since the advance trigger output did not appear to work. If that delay were removed, the execution time would have been about 3 ms, which is consistent with normal DAQ modes.*

3 *The VXI SCPI numbers were gathered with an E1406A GPIB Command Module*

4 *The SCXI-1128 was unable to generate a trigger back to the DMM, so a full handshake is not possible. Thus the SCXI measurement is a synchronous type, in which the DMM takes 20 readings using a sample interval, and its Measurement Complete signal is used to advance the scanning to the subsequent channel. Therefore the execution time depended upon the DMM sample interval. The reported number is the execution time for a sample interval of 0.0, representing the overhead. As sample interval goes up, the reported time equals the sample interval.*

N/P *Not possible. PXI instruments require the use of drivers.*

There are a few interesting nuggets of information in this data. As mentioned earlier, Data Acquisition mode results in much faster execution time, sometimes by as much as an order of magnitude. Also, there are wide variations in execution times; Keithley's use of fast reed relays didn't help much, for example, since the instrument was so slow at taking readings. PXI was fastest, but its performance, which was only measured using FET switches, was comparable to the 34980A FET switches. Its execution time would increase by basically the switching time of reeds or armatures if they were used instead. Also, depending upon how the driver is implemented, it can be faster than the SCPI equivalent by taking advantage of state-caching.

Product breadth

The following charts show the types of modules that are available for the various platforms.

	Agilent 34980A	Agilent 34970A	Agilent 3499A	Agilent VXI	NI PXI/SCXI	Keithley 2701	Racal 1256
Matrix relay							
FET	None	None	None	None	4x6 2w 4x8 2w	None	None
Reed	2, 4x8 2w	None	None	4x32 2w	4x16 2w 4x32 2w 4x64 2w	None	None
Armature	2, 4x8 2w 2, 4x16 2w	4x8 2w	4x4 2w 4x8 2w	16x16 4x16 8x32 8x8 4x16 2w	4x6 2w 8x16 2w	6x8 2w	12x12 2w (7 configurations)
Mux relay							
FET	40-ch 2w	None	8-ch 2, 4-ch 4, 2-ch se	16-ch 3w 32-ch 16-ch 3w	24-ch 2w 32-ch 2w	20-ch	None
Reed	40-ch 2w	20-ch 2w	40-ch se	16-ch 3w 48-ch se 64-ch 3w 256-ch	64-ch 2w 128 2w 256 2w	32-ch 2w	42-ch (dry or merc wetted)
Armature	40-ch 2w 27-ch 2w	20-ch 2w 40-ch 1w	10-ch 2w 20-ch-2w 40-ch 2w	64-ch 2w		20-ch 2w 32-ch 2w 40-ch 2w	23-ch 2w 16, 1x4 8, 8-ch 2w
RF	4, 1x4 50 or 75Ω	2, 1x4 50 or 75Ω	2, 1x4 2, 1x6 1x9 50 or 75Ω	2, 1x4 6, 1x4 50 or 75Ω	4, 1x4 8, 1x4 1x16 1x32 50Ω	2, 1x4	10, 1x4 2, 1x4 1x6 2, 1x6
GP relay							
Armature	28-C/4-A, 20-ch A	20-ch C	7/8/10/20-ch C, 40-ch A	16-ch C 32-ch C 40-ch A 64-ch A	8/16/32/40 C 16/31/100 A	40-ch A	12 A 12 B 12 C 20 DPDT 52 C 80 A, low/hi power versions
RF	2-ch SPDT, 3-ch SPDT		3-ch SPDT, relay driver	18 GHz, 3-ch SPDT, relay driver	32, 64 relay drivers	None	20-ch SPDT 50/75 Ω 2 SPDT 5 SPDT 2x2 xfer switches

Continued on Page 10

Key:

A	Form A
B	Form B
C	Form C
1w	1-wire
2w	2-wire

Product breadth *continued*

	Agilent 34980A	Agilent 34970A	Agilent 3499A	Agilent VXI	NI PXI/SCXI	Keithley 2701	Racal 1256
DMM	6.5 digit built-in	6.5 digit built-in	External	5.5 digit or 6.5 digit	5.5 digit or 6.5 digit	6.5 digit built-in	External
Digital I/O	64-ch	On multifunction card	16-bit, 32-bit TTL	4, 8-bit 72-ch out 96-ch dio 64-ch iso in	11 different cards	On multifunction cards	48 oc, 96 (ttl, cmos, oc)
DAC	4-ch iso waveform	On multifunction card	On multifunction cards	4-ch, 8/16-ch		On multifunction card	None
Counter/totalizer	On DIO card	On multifunction card	On multifunction cards	E1333A: 2-ch @ 100 MHz + 1-ch @ 1 GHz	4- or 8-ch to 125 MHz	On multifunction card	None
Multifunction	32 dio 2 ±12V DAC 100 KHz Totalizer	16-bit dio, 26-bit counter, 2, 16-bit dac	15 gp/16 dio 4x4 mat/16 dio 2 DAC/16 dio	M-modules	digitizer, counter, dio, dac	20-ch 2w mux, 2DAC, 16 dio, 1 counter 10-ch 2w mux, 32 dio	None
Analog bus	Internal 4Bus 2w	None	None	None	PXI: none SCXI: 3Bus 2w	DMM Hi/Lo	DMM Hi/Lo
Other	Breadboard	None	None	Scopes, arbs, etc	Scopes, digitizers, arbs, etc	None	Breadboard

Cost of ownership

Prices of several Switch/Measure solutions are shown in the table at right. Those requiring separate DMM and switchbox were eliminated. PXI and VXI suffer from the need to pay for an expensive cardcage that was meant for high-speed instrumentation. One must also pay for an interface card on both the computer end and the cardcage end. This creates an overhead that must be added to the cost of every slot used. In addition, it can take considerably longer to get a solution implemented using discrete cards than by using a Switch/Measure box. See the next section for a more thorough discussion of this. Using VB.NET, it took a few minutes to implement a scanned measurement on the 34980A and 34970A Switch/Measure Units. In contrast, it took two weeks and several support calls to do the same measurement using PXI and SCXI.

Instrument	**List price (USD, as of September 1, 2004)**		
Agilent 34980A	$2350	incl. DMM, LAN, USB, GPIB, 8 slots	**($294/slot)**
Agilent 34970A	$1477	incl. DMM, GPIB, RS232, 3 slots	**($492/slot)**
Keithley 2750	$2995	incl. DMM, GPIB, RS232, 5 slots	**($600/slot)**
NI PXI	$5485	PXI-1042 8-slot cage ($1995), 6 slots avail. PXI-4070 DMM ($1995), (uses 1 slot) MXI-4 interface (PXI-PCI8331) ($1495), (uses 1 slot)	**($914/slot)**

Cards for the Agilent 34980A range from $495 to $2000.

Cards for the Agilent 34970A range from $340 to $501.

Cards for the Keithley 2701/2750 range from $445 to $995, plus one microwave module at $1995

Switch cards for PXI from NI range from $495 to $1995, with one high density card at $4795.

From nearly any perspective, the 34980A shines. It holds more cards than many other solutions, has a wide selection of cards covering higher frequency ranges, has more computer interface options, and has the lowest price per slot. It also has support for more programming environments (via LabVIEW and IVI-COM drivers).

Ease of Use issues

Numerous problems were encountered over the course of the 3 week evaluation period, which are summarized here:

1. **Keithley Firmware update required.** If the program aborts or is ended without executing a "close" on the Keithley 2701, that instrument cannot be used again until power is cycled. The Keithley web site FAQ has an item that says that version A06 of their firmware allows a second port (2701) to be opened in order to send the command "KI2701" which will reset port 1394 so it can be used again. The unit used for the testing had version A04, so A06 was downloaded from their web site and the unit was re-flashed. No problems were encountered doing this and it indeed fixed the problem.

2. **Unusual behavior.** The Keithley 2701 was found to be resetting the DMM aperture time to SLOW (5 PLC) whenever a ROUT:MULT:CLOS (Close multiple relays) command was sent. This necessitated the addition of a command to reset the aperture time to .01 NPLC. This was also necessary in DAQ mode even though the Configure statement sets the resolution. No other instruments behaved this way, so it took a while to understand why the readings were so slow.

3. **NI-VISA setup changes required.** If NI VISA is used, one must also run MAX (Measurement Automation Explorer) and enable the "Passport to Tulip" interface driver in the Tools->NI-VISA->VISA Options->Passports or none of the Agilent VXI/GPIB instruments will be recognized. However, with NI Measurement Studio installed, this interface could not be used because the NI Passport interface driver calls AgVisa32.dll and it caused an exception upon exit. A support session was initiated with NI that took a week to resolve. NI's Tulip passport driver writer said this is a known problem. The fix was to install a patched version of NIVisaTulip.dll. This fixed the problem, but we observed that the latest NI install disks are still using the older version.

4. **NI IVI-compliance problems.** Agilent's Driver Wrapper Wizard can only wrap IVI-C and VXIplug&play drivers that are properly installed according the IVI Foundation Specs. NI-DMM and possibly other NI IVI-C drivers did not install correctly unless the "LabWindows/CVI examples" box was checked during the installation step. At the time of this evaluation, NI did not provide .NET compatibility for their instruments, but they did offer .NET-wrapped code that can be downloaded from their web site and manually attached to a program. This was done for the NI-DMM driver. However, NI chose a Namespace of "InstrumentDriverInterop" in that wrapper, which conflicted with the one created by Agilent Toolkit. The NI Namespace was renamed "InstDvrInterop" to fix the conflict. An alternative is to explicitly spell out the namespace every time it is used instead of using shortcuts. For example, if one specifies "Imports Agilent.TMFramework" at the beginning of the program, all the subcategories in that framework can be used without using that prefix. One such category is InstrumentDriverInterop. So, one can either specify,

   ```
   Agilent.TMFramework.InstrumentDriverInterop.xxx
   ```

   or simply,

   ```
   InstrumentDriverInterop.xxx
   ```

   (where xxx is the next level of functionality). However, when one has attached the NI-wrapped driver with the same namespace, the conflict must be resolved.

5. **Insufficient documentation.** NI's examples for use of their PXI-4070 DMM were in VB6 and VC++ and LabVIEW. Their examples for DAQmx Switches in .NET do not include hardware triggers, only software triggers. The hardware triggers require use of the backplane trigger busses, but the parameters that use these require strings, and there are no examples of what strings to use. The IntelliSense help that pops up tried to run a javascript to give help, but it did not work. Running Start->Programs->NationalInstruments->NI-DAQ->DAQmx Documentation manually retrieved the intended help. However, when using the names that it gave for trigger busses in the code that they required, none could be found that worked. This took a lot of trial and error. Various names that were tried included PXITRig0, LBR_Trig0, TTL0 and

NI_VAL_TRIG_TTL0. A LabVIEW example was then loaded in order to see what they were using. It revealed that the new naming conventions were required, which are all path based—"/SC1Mod3/TrigIn", for example. However, none of the names in the LabVIEW example worked. Another support session was initiated, and the response finally came back that the SCXI-1128 does not support handshaking, only synchronous mode (unidirectional triggering) and that either a newer card should be used, or Traditional DAQ should be used instead. We asked NI how we could have known this ahead of time. The response was, *"Unfortunately, the triggering limitation is one of those things that would have been hard to find out without asking someone or thoroughly reading the help manual before making the purchase."*

6. **PC shutdown required when swapping cards in PXI.** Whenever a card is moved in a PXI cardcage, power must be cycled, but that can't be done with the PC power on since the PXI backplane is an extension of the PCI bus in the computer. Thus a PC reboot is required, which is time-consuming.

7. **Version conflicts.** Many version conflicts were encountered. For example, while troubleshooting SCXI switching problems, NI-DAQ 7.1 was installed on a Measurement Studio 7.0 installation. This caused numerous problems that were only solved by uninstalling all NI software and reinstalling it. This took the better part of a day.

8. **How can Agilent and NI hardware be controlled from one program?** NI MAX can find devices on all NI interfaces, but cannot directly control Agilent interfaces, such as the FireWire interface to VXI, USB/GPIB converter or the PCI GPIB card. Agilent Instrument Explorer cannot currently find PXI devices. How can this best be resolved in a test system that needs to communicate with devices from both vendors? The solution was to install Agilent I/O libraries in a "side-by-side" mode (described in the I/O Libraries Help file), then to enable the Passport-Tulip interface driver in NI MAX as described earlier. This causes VISA calls to those interfaces to be routed from NI VISA to Agilent VISA, which then controls the relevant Agilent interfaces while still allowing NI interfaces such as MXI-3 to work directly through NI VISA and Passport drivers.

Code Examples

Here are the programming requirements in the Visual Basic.NET environment for a simple EFT (close/measure/open) and DAQ (scanned) measurement using the DMM in DC Volts on the 10V range, with a 20-channel mux.

There are several things that can be observed by looking at the code:

1. Switch/measure units with internal DMMs take a lot of the work out of DAQ measurements. This is because the triggering functions are done for you. It literally took only a few minutes to create working code with these instruments. It took 2 weeks to get the PXI/SCXI measurement to work, largely because of difficulty understanding the triggering requirements.
2. SCPI strings can be concatenated, making long strings. This saves a little execution time because there is no extra overhead in multiple function calls. However, as noted earlier, if it is not necessary to send the string in the first place, the command parsing time in a slow instrument can easily dwarf the function call execution time. If high throughput is a requirement in your application, you should spend time evaluating state-caching drivers versus SCPI.
3. IVI-C (with .NET wrappers), IVI-COM and VXIplug&play drivers are all very similar in usage in the .NET environment. All provide various degrees of IntelliSense help. Not obvious in the listings below is the fact that IVI-COM help is much more useful than the other two. The PXI/SCXI help was not adequate, requiring frequent reference to the on-line manuals and several e-mail support sessions to NI.

All VB.NET programs that use Agilent T&M Toolkit automatically insert the following:

```
Imports Agilent.TMFramework
Imports Agilent.TMFramework.DataAnalysis
Imports Agilent.TMFramework.DataVisualization
Imports Agilent.TMFramework.InstrumentIO
Imports Agilent.TMFramework.InstrumentDriverInterop
```

When using NI Measurement Studio, the following must be manually added:

```
Imports NationalInstruments
Imports niDMM_32.NIDMMMeasurementConstants
```

Declarations are added automatically by Toolkit to the “Public Class”:

VXIplug&play Driver declarations:

```
Dim myHpe1476 As InstrumentDriverInterop.VxipnpWrappers.Hpe1476
Dim myHpe1411 As InstrumentDriverInterop.VxipnpWrappers.Hpe1411
Dim myKe2700 As InstrumentDriverInterop.VxipnpWrappers.Ke2700
Dim myRi1256 As InstrumentDriverInterop.VxipnpWrappers.Ri1256
```

IVI-C Driver declarations wrapped by Agilent Toolkit Driver Wrapper Wizard

```
Dim myniSwitch As InstrumentDriverInterop.IviCWrappers.NiSwitch
```

Direct I/O (SCPI) declarations

```
Dim myDSCPI As InstrumentIO.DirectIO
Dim my34980 As InstrumentIO.DirectIO
Dim my34970 As InstrumentIO.DirectIO
Dim my3499 As InstrumentIO.DirectIO
Dim my34401 As InstrumentIO.DirectIO
```

IVI-COM driver declarations

```
Dim myAgilent34401 As Agilent.Agilent34401.Interop.Agilent34401
Dim myAgilent34970 As Agilent.Agilent34970.Interop.Agilent34970
Dim myAgilent3499 As Agilent.Agilent3499.Interop.Agilent3499
```

IVI-C declarations added manually when using NI Measurement Studio

```
Dim PXIDMM As InstDvrInterop.Ivi.niDMM
Dim SCXI As InstDvrInterop.Ivi.niSwitch
```

This is how instruments are initialized:

VXIplug&play Driver initialization for the E1411/E1476 combo:

Separate DMM usage:

```
myHpe1411 = New InstrumentDriverInterop.VxipnpWrappers.Hpe1411("VXI0::24::INSTR", True, True)
```

Combined DMM/Switch usage:

```
myHpe1411and1476 = New InstrumentDriverInterop.VxipnpWrappers.Hpe1411("VXI0::(24,25)::INSTR", True, True)
```

D-SCPI session for E1411/E1476 VXI using GPIB command module:

```
myDSCPI = New InstrumentIO.DirectIO("GPIB1::9::3::INSTR", False)
```

VXIplug&play (Wrapped) Driver session for Ke2701:

```
myKe2700 = New VxipnpWrappers.Ke2700("TCPIP0::169.254.105.002::1394::SOCKET", False, False)
myKe2700.Reset()
```

IVI-COM Driver session for 3499:

```
myAgilent3499 = New Agilent.Agilent3499.Interop.Agilent3499Class
myAgilent3499.Initialize("GPIB0::9::INSTR", True, True, Nothing)
myAgilent3499.Utility.Reset()
```

SCPI session for 3499:

```
my3499 = New InstrumentIO.DirectIO("GPIB0::9::INSTR", False, False)
my3499.Timeout = 2000
```

IVI-COM driver setup for 34401:

```
myAgilent34401 = New Agilent.Agilent34401.Interop.Agilent34401Class
myAgilent34401.Initialize("GPIB0::22::INSTR", True, True, Nothing)
```

SCPI session for 34401:

```
my34401 = New InstrumentIO.DirectIO("GPIB0::22::INSTR", False, False)
my34401.Timeout = 2000
```

VXIplug&play (Wrapped) Driver session for Racal 1256:

```
myRi1256 = New InstrumentDriverInterop.VxipnpWrappers.Ri1256("GPIB0::14::INSTR", True, True)
```

IVI-COM driver session for 34970:

```
myAgilent34970 = New Agilent.Agilent34970.Interop.Agilent34970Class
myAgilent34970.Initialize("GPIB0::8::INSTR", True, True, Nothing)
```

SCPI session for 34970:

```
my34970 = New InstrumentIO.DirectIO("GPIB0::8::INSTR", False, False)
```

IVI-COM Driver session for 34980:

```
myAgilent34980 = New Agilent.Agilent34980.Interop.Agilent34980Class
myAgilent34980.Initialize("TCPIP0::169.254.9.80::INSTR ", True, True, Nothing)
myAgilent34980.Reset
```

SCPI session for 34980:

```
my34980 = New InstrumentIO.DirectIO("TCPIP0::169.254.9.80::INSTR")
my34980.WriteLine("*RST")
```

IVI-C Driver session for PXI/SCXI:

```
PXIDMM = New InstDvrInterop.Ivi.niDMM("DAQ::8::INSTR", True, True)
myniSwitch = New IviCWrappers.NiSwitch("SCXI1::3", False, True)
```

Keithley 2701 example

```
Dim index As Integer
Dim rdgArray(80) As Double
Dim reading As Double
Dim numPts As Integer

myKe2700.ConfigureAutoZeroMode(VxipnpWrappers.Ke2700.AutoZeroModeEnum.Off)
myKe2700.ConfigureMeasurement(VxipnpWrappers.Ke2700.MeasFunctionEnum.ValDcVolts, 10.0, 0.001)
```

EFT Mode (Close/Measure/Open):

```
For index = 1 To numChannels
    myKe2700.ConfigureSwitches( _
s101, _
VxipnpWrappers.Ke2700.SwitchModeEnum.ValCloseSingleOpenOtherChannels)
    myKe2700.ConfigureApertureTimeInfo(.01, VxipnpWrappers.Ke2700.ApertureTimeUnitEnum.Nplc)
    myKe2700.Read(5000, reading)
    myKe2700.ConfigureSwitches( _
s101, _
VxipnpWrappers.Ke2700.SwitchModeEnum.ValOpenMultiple)
Next
```

DAQ Mode (Scanned):

```
myKe2700.SetChannelList(s101120)
myKe2700.ConfigureMultiPoint( _
    1, _
    numChannels, _
    myKe2700.TriggerSourceEnum.Immediate, _
    0.0)
myKe2700.ConfigureApertureTimeInfo(.01, VxipnpWrappers.Ke2700.ApertureTimeUnitEnum.Nplc)
myKe2700.ReadMultiPoint(5000, 80, rdgArray, numPts)
```

RACAL 1256/Ext. DMM example

```
Dim index As Integer
Dim readings(20) As Double
Dim trigDelay As Double = 0.02

myHpe1411.CalZeroAuto(False)

myHpe1411.VoltDcRang(False, _
    VxipnpWrappers.Hpe1411.RangeEnum2.VoltDcRang64V)

myHpe1411.VoltDcRes(VxipnpWrappers.Hpe1411.VoltDcResEnum.VoltRes488Micro)

myHpe1411.Trigger(1, False, trigDelay, VxipnpWrappers.Hpe1411.SourceEnum1.Immediate)

myHpe1411.Sample(1, VxipnpWrappers.Hpe1411.SourceEnum.Immediate, 0.0)
```

EFT Mode (Close/Measure/Open):

```
For index = 1 To numChannels

    myRi1256.OperateSingle138( _
        x.ModuleAddressEnum1._1, _
        x.OperationEnum.Close, _
         x.RelayTypeEnum.Mux0, _
         1)

  myHpe1411.ReadQ(readings, 80)

  myRi1256.OperateSingle138( _
         x.ModuleAddressEnum1._1, _
         x.OperationEnum.Open, _
         x.RelayTypeEnum.Mux0, _
         1)
Next
```

DAQ Mode (Scanned):

```
myHpe1411.Trigger(1, False, trigDelay,    VxipnpWrappers.Hpe1411.SourceEnum1.Immediate)

myHpe1411.Sample(numChannels, VxipnpWrappers.Hpe1411.SourceEnum.Timer, trigDelay)

myRi1256.ConfigOutputTrigState(VxipnpWrappers.Ri1256.OutputTrigStateEnum.Off)

myRi1256.ConfigInputTrigSource(VxipnpWrappers.Ri1256.TriggerSourceEnum1.TrigExt)

myRi1256.ArmTrigger(VxipnpWrappers.Ri1256.ArmTypeEnum.Cont)

myRi1256.DefScanList(New System.Text.StringBuilder("1(0:7,10:17,20:23)"))

myRi1256.TriggerImmediate() ' go to the first relay in the scanlist

myHpe1411.ReadQ(readings, numChannels)
```

Agilent 34980A example using driver

```
Dim index As Integer
Dim rdgArray(80) As Double
```

EFT Mode (Close/Measure/Open):

```
myAgilent34980A.Scan.ScanList = ""

myAgilent34980A.Voltage.DCVoltage.Configure("", 10.0,
Agilent.Agilent34980A.Interop.Agilent34980AResolutionEnum.Agilent34980AResolutionLeast)

myAgilent34980A.Voltage.DCVoltage.AutoZero("") =
Agilent.Agilent34980A.Interop.Agilent34980AAutoZeroEnum.Agilent34980AAutoZeroONCE

myAgilent34980A.Display.DisplayEnabled = False

myAgilent34980A.Trigger.Configure(
Agilent.Agilent34980A.Interop.Agilent34980ATriggerSourceEnum.Agilent34980ATriggerSourceImmediate, 1, 0, 1)

For index = 1 To numChannels
        myAgilent34980A.Route.Close("1001")
        myAgilent34980A.Measurement.Initiate()
        rdgArray = myAgilent34980A.Measurement.FetchNumbersOnly
        myAgilent34980A.Route.Open("1001")
Next
```

DAQ Mode (Scanned):

```
myAgilent34980A.Scan.ScanList = "1001:1020"

myAgilent34980A.Voltage.DCVoltage.Configure("1001:1020", 10.0,
Agilent.Agilent34980A.Interop.Agilent34980AResolutionEnum.Agilent34980AResolutionLeast)

myAgilent34980A.Voltage.DCVoltage.AutoZero("1001:1020")
Agilent.Agilent34980A.Interop.Agilent34980AAutoZeroEnum.Agilent34980AAutoZeroONCE

myAgilent34980A.Display.DisplayEnabled = False

myAgilent34980A.Trigger.Configure(
Agilent.Agilent34980A.Interop.Agilent34980ATriggerSourceEnum.Agilent34980ATriggerSourceImmediate, 1, 0, 1)

myAgilent34980A.Route.Delay("1001:1020") = 0

myAgilent34980A.Measurement.Initiate()

rdgArray = myAgilent34980A.Measurement.FetchNumbersOnly
```

Agilent 34970A example using driver

```
Dim index As Integer
Dim readings(80) As Double
Dim rdgs(80) As String

myAgilent34970.Display.DisplayEnabled = False

myAgilent34970.Voltage.AutoZero(101) =
Agilent.Agilent34970.Interop.Agilent34970AutoZeroEnum.Agilent34970AutoZeroONCE
```

EFT Mode (Close/Measure/Open):

```
myAgilent34970.Voltage.DCVoltage.Configure("101:101", 10, 0.001)

For index = 1 To numChannels
  rdgs = myAgilent34970.Scan.Read()

Next
```

DAQ Mode (Scanned):

```
myAgilent34970.Voltage.DCVoltage.Configure("101:120", 10, 0.001)

rdgs = myAgilent34970.Scan.Read()
```

Agilent 3499A example using driver

```
Dim index As Integer
Dim readings(80) As Double
Dim numRdgs As Integer

myAgilent34401.Advanced.AutoZero = _
            Agilent.Agilent34401.Interop.Agilent34401AutoZeroEnum.Agilent34401AutoZeroOnce

myAgilent34401.Display.Enabled = False

myAgilent34401.DCVoltage.Configure(10.0, 0.001)
```

EFT Mode (Close/Measure/Open):

```
myAgilent34401.Trigger.Source =
   Agilent.Agilent34401.Interop.Agilent34401TriggerSourceEnum.Agilent34401TriggerSourceImmediate

For index = 1 To numChannels
  myAgilent3499.Route.Close("@200")
  readings(0) = myAgilent34401.Measurement.Read(5000)
  myAgilent3499.Route.Open("@200")
Next
```

DAQ Mode (Scanned):

```
myAgilent3499.Configure.ExtTriggerSource = 0
myAgilent3499.Configure.ExtTriggerOutput = True

myAgilent3499.Scan.ArmSource = _
   Agilent.Agilent3499.Interop.Agilent3499SourceEnum.Agilent3499SourceImmediate

myAgilent3499.Scan.ArmCount = 1

myAgilent3499.Scan.TriggerSource = _
   Agilent.Agilent3499.Interop.Agilent3499SourceEnum.Agilent3499SourceMIX

myAgilent3499.Configure.MuxFunction(2) = _
   Agilent.Agilent3499.Interop.Agilent3499WireEnum.Agilent3499WireWire2

myAgilent3499.Scan.ScanList = "@200:219"

myAgilent34401.Trigger.Source = _
   Agilent.Agilent34401.Interop.Agilent34401TriggerSourceEnum.Agilent34401TriggerSourceExternal

myAgilent34401.Trigger.MultiPoint.Count = numChannels

myAgilent34401.System.WaitForOperationComplete(5000)

myAgilent34401.Measurement.Initiate()

myAgilent3499.Scan.Initiate()

myAgilent3499.System.IO.WriteString("*TRG", True)

myAgilent3499.System.WaitForOperationComplete(5000)

myAgilent34401.Measurement.FetchMultiPoint(5000, readings)
```

Agilent E1411B/E1476A example using driver

```
Dim index As Integer
Dim opc As Short
Dim readings(80) As Double

myHpe1411and1476.CalZeroAuto(False)

myHpe1411and1476.VoltDcRang(False, VxipnpWrappers.Hpe1411.RangeEnum2.VoltDcRang64V)

myHpe1411and1476.VoltDcRes(VxipnpWrappers.Hpe1411.VoltDcResEnum.VoltRes488Micro)
```

EFT Mode (Close/Measure/Open):

```
For index = 1 To numChannels
    myHpe1476.ClosCardChan(1, 0)
    myHpe1411.ReadQ(readings, 80)
    myHpe1476.OpenCardChan(1, 0)

Next
```

DAQ Mode (Scanned):

```
myHpe1411and1476.ConfigureList( VxipnpWrappers.Hpe1411.FuncEnum1.ConfListVoltDc, "100:119")

myHpe1411and1476.InitImm()

myHpe1411and1476.TimedFetchQ(5000, readings, 20)
```

NI PXI-4070/SCXI-1128 example

```
Dim index As Integer
Dim readings(80) As Double
Dim numRdgs As Integer
Dim SampInt as Double
```

EFT Mode (Close/Measure/Open):

```
PXIDMM.ConfigureAutoZeroMode(NIDMM_VAL_AUTO_ZERO_ONCE)

PXIDMM.ConfigureMeasurement(NIDMM_VAL_DC_VOLTS, 10.0, 0.001)

PXIDMM.ConfigureTrigger(NIDMM_VAL_IMMEDIATE, 0.0)

For index = 1 To numChannels
    myniSwitch.Connect("ch0", "com0")
    PXIDMM.Initiate()
    PXIDMM.Fetch(5000, readings(0))
    myniSwitch.Disconnect("ch0", "com0")

Next
```

DAQ Mode (Scanned):

```
PXIDMM.ConfigureAutoZeroMode(NIDMM_VAL_AUTO_ZERO_ONCE)

PXIDMM.ConfigureMeasurement(NIDMM_VAL_DC_VOLTS, 10.0, 0.001)

PXIDMM.ConfigureMeasCompleteDest(NIDMM_VAL_LBR_TRIG_0)

PXIDMM.ConfigureMultiPoint(1, numChannels, NIDMM_VAL_INTERVAL, SampInt)

myniSwitch.ConfigureScanList("sc1!md3!ch0:19->com0;", _
    IviCWrappers.NiSwitch.ScanModeEnum.BreakBeforeMake)

myniSwitch.ConfigureScanTrigger(0.0, _
    IviCWrappers.NiSwitch.TriggerInputEnum.Ttl0, _
    IviCWrappers.NiSwitch.ScanAdvancedOutputEnum.None)

myniSwitch.InitiateScan()

PXIDMM.Initiate()

PXIDMM.FetchMultiPoint(5000, numChannels, readings, numRdgs)
```

Agilent 34980A example using SCPI

```
Dim index As Integer
Dim readings As String

my34980.WriteLine("Rout:Scan (@)")

my34980.WriteLine("CONF:VOLT:DC 10,.001")

my34980.WriteLine("ZERO:AUTO ONCE;:DISP OFF;:TRIG:DELAY 0;:TRIG:SOUR IMM;:TRIG:COUN 1;:SAMP:COUN 1")
```

EFT Mode (Close/Measure/Open):

```
For index = 1 To numChannels
    my34980.WriteLine("ROUT:CLOS (@6001)")
    my34980.WriteLine("INIT;FETC?")
    readings = my34980.Read()
    my34980.WriteLine("ROUT:OPEN (@6001)")

Next
```

DAQ Mode (Scanned):

```
my34980.WriteLine("Rout:Scan (@6001:6020)")

my34980.WriteLine("Conf:volt:dc 10,.001,(@6001:6020)")

my34980.WriteLine("ZERO:AUTO ONCE;:DISP OFF;:TRIG:DELAY 0;:TRIG:SOUR IMM;:TRIG:COUN 1;:SAMP:COUN 1")

my34980.WriteLine("INIT;FETC?")

readings = my34980.Read()
```

Agilent 34970A example using SCPI

```
Dim index As Integer
Dim Readings As String
Dim Points As Integer
Dim replyString As String

my34970.WriteLine("DISP OFF;: TRIG:COUN 1;SOUR IMM")
```

EFT Mode (Close/Measure/Open):

```
my34970.WriteLine("Conf:volt:dc 10,.001,(@101:101)")

For index = 1 To numChannels
    my34970.WriteLine("INIT;FETC?")
    Readings = my34970.Read()

Next
```

DAQ Mode (Scanned):

```
my34970.WriteLine("Conf:volt:dc 10,.001,(@101:120)")

my34970.WriteLine("INIT;FETC?")

Readings = my34970.Read()
```

Agilent 3499A/34401A example using SCPI

```
Dim index As Integer
Dim Readings As String
Dim dummy As String

my34401.WriteLine("DISP OFF; :ZERO:AUTO ONCE;:CONF:VOLT:DC 10,.001")
```

EFT Mode (Close/Measure/Open):

```
my34401.WriteLine("TRIG:SOUR IMM;COUN 1;:SAMP:COUN 1")

For index = 1 To numChannels
    my3499.WriteLine("ROUT:CLOS (@200)")
    my34401.WriteLine("INIT;FETC?")
    Readings = my34401.Read()
    my3499.WriteLine("ROUT:OPEN (@200)")

Next
```

DAQ Mode (Scanned):

```
my3499.WriteLine("CONF:EXT:SOUR 0;OUTP 1")

my3499.WriteLine("ARM:SOUR IMM;COUN 1")

my3499.WriteLine("TRIG:SOUR MIX")

my3499.WriteLine("FUNC 2,2") ' 2-wire mode

my3499.WriteLine("SCAN (@200:219)")

my34401.WriteLine("TRIG:SOUR EXT;COUN " & Str(numChannels))

my34401.WriteLine("*OPC?")

dummy = my34401.Read

my34401.WriteLine("INIT")

my3499.WriteLine("INIT")

my3499.WriteLine("*TRG")

my3499.WriteLine("*OPC?")

dummy = my3499.Read

my34401.WriteLine("FETC?")

Readings = my34401.Read
```

Agilent VXI E1411B/E1476A example using D-SCPI

```
Dim index As Integer
Dim Readings As String

myDSCPI.WriteLine("ZERO:AUTO ONCE;: TRIG:SOUR IMM;DELAY 0")
```

EFT Mode (Close/Measure/Open):

```
myDSCPI.WriteLine("CONF:VOLT:DC 10,.001 (@100)")

myDSCPI.WriteLine("TRIG:COUN 1")

myDSCPI.WriteLine("SAMP:COUN 1")

For index = 1 To numChannels
    myDSCPI.WriteLine("INIT; FETC?")
    Readings = myDSCPI.Read()

Next
```

DAQ Mode (Scanned):

```
myDSCPI.WriteLine("CONF:VOLT:DC 10,.001 (@100:119)")

myDSCPI.WriteLine("INIT;FETC?")
Readings = myDSCPI.Read()
```

© Agilent Technologies, Inc. 2005
Printed in USA June 14, 2005
5989-1929EN

Agilent Technologies