IVI Introduction

Autotestcon 2016
What is IVI?

The primary purpose of the Consortium is to:

• Promote the development and adoption of standard specifications for programming test instrument

• Focus on the needs of the people that use and develop test systems who must take off-the-shelf instrument drivers and build and maintain high-performance test systems

• Build on existing industry standards to deliver specifications that simplify interchanging instruments and provide for better performing and more easily maintainable programs

From IVI Foundation *By-laws*
IVI fit with other specs

- **SCPI**
- **IVI 4.x (Classes)**
  - Scope
  - DMM
  - FG
  - DC
  - AC
  - Switch
  - Spec.
  - SW
  - Counter
  - DownConv
  - UnConv
  - Digitiz
- **IVI 3.x (Arch)**
  - C
  - COM
  - .NET
- **VISA**
  - C/COM/,.NET
  - Message
  - Register
- **VXI plug&play**
- **IVI 6.3 PXI plug-in**
- **PXI-2 and PXI-6: Software**
- **AXIe 2.0**
- **AXIe 3.1**
- **T&M Specific Protocols/Standards**
- **Physical Connection**
- **IO Interfaces & SW Protocols**
- **Instrument Capabilities**
  - Programming Interfaces for C/C++/C#/VB
  - LabVIEW, etc
• SCPI provided necessary standards based on GPIB needs
  - Command strings natural match to GPIB
  - HiSLIP allows connection to LXI

• VXIplug&play added drivers necessary for VXI
  - Used with other I/O to provide necessary driver

• IVI enhances VXIplug&play with new features & support for current tools
Comparing Drivers and SCPI

• Programming with SCPI
  viPrintf(vi, “MEAS:VOLT? %f, %f”, range, resolution);
  viScanf(vi, &reading);
  – Program deals with strings sent to/from the instrument
  – Syntax errors caught by instrument when program is run
  – Checking for errors requires another sequence to read error
  – Simple model that requires no driver install

• Programming with IVI (IVI-C here)
  Ag34410_MeasureDCVolt(vi, range, resolution, &reading);
  – Program variables sent directly – no chance for SCPI syntax errors
  – Syntax errors caught by compiler or editor
  – No performance impact due to string manipulation
  – Uses debug tools and techniques the programmers knows
What are IVI Drivers – Really??

- Architecture specifications
- Instrument class specifications
- A library of shared software components

13 specs @ ~220 pages

~1140 pages of specs
The IVI Architectures

IVI Provides: C, COM, and .NET
• C dll for environments that use DLLs
• COM Components for COM and .NET ADEs
• .NET Assemblies for .NET ADEs

Architectures make use of same class definition
Architectures have specific rules for installation, style, etc.

Details in next section
IVI Shared Components

IVI Provides several common components to enable multi-vendor systems (more information in the final section)

- C Shared Components
- Floating Point Services
- IVI-COM Session Factory
- Configuration Server
- COM Type Libraries
- .NET PIAs
- .NET Shared Components
What is IVI Compliant -Really??

**IVI Compliant**
- Common behavior model
- Support for IVI Features
  - Simulation, IO, doc, ....
- Standard install
- Common API for common tasks
  - ~40 common functions
  - Simulation, Caching, Open, Close, Initialize, SW Trigger, Status check, Version ....

**Class Compliant**
- Instrument Class API
- Custom API still available
  - Especially for capabilities beyond the class
- Simplifies exchanging instruments

• Consistent API
  - Common organization, data types, naming
Why IVI? – Simpler to use

Uniform way of doing common tasks

- Instantiation, initialization, shutdown
- Controlling driver features – state caching, error query, simulation, etc.
- Configuration and installation
  - Fixed locations for binaries, source, headers, documentation, examples
  - Proper registry entries always made
  - Common protocol to open/close (standard I/O address is a big benefit)
  - Consistent solution for managing driver versions
- Standard mechanism for handling multi-channel devices
  - aka repeated capabilities in IVI parlance
- Standard error reporting
Why IVI? – Common Features

Key Capabilities that simplify program development

- Syntactic Interchangeability
- Simulation
- Fine grained control through properties
- Usable in many ADEs
- Documentation of SCPI commands used by function
- DirectIO (drivers provide access to SCPI)
- Attributes for all parameters (fine grained control)
- Buildable source for message based instruments (SCPI)
- Tested using a IVI-defined process
Why IVI? – One Driver for any ADE

• IVI Drivers (C/COM/.NET) provide a first class experience in *nearly any ADE*
  – Visual Basic 6
  – Visual C++
  – Visual C# and Visual Basic.NET
  – VBA (Excel, Word, PowerPoint)
  – LabVIEW
  – LabWindows/CVI
  – MATLAB
  – Agilent VEE
IVI Registration Page

- IVI maintains a registration database
- IVI requires that drivers claiming compliance be registered
- For users:
  - List of drivers, supported instruments
  - Mechanism to address defects
Motivations for IVI.NET

- Present an API optimized for .NET
- Simplify driver source code
  - Allow end users to understand instrument behavior by examining driver source
  - Allow end users to modify and add features to drivers on their own
- Richer, more expressive APIs
  - More flexibility with API data types
  - Clean handling of asynchronous notifications (aka “events”)
  - Simplified error Handling
- Side-by-side deployment of drivers
  - Only one version of an IVI-COM or IVI-C driver can be installed at a time
  - IVI.NET allows multiple versions of a driver to be installed
Richer Type System

Both IVI-COM and IVI-C drivers use a limited set of data types
- Integers, floats, Booleans, strings
- Arrays of the above, but extra parameters are required in IVI-C

IVI-C cannot expose an array of strings

IVI-C cannot expose structs
- Can be done in IVI-COM, but it’s tedious to implement

```c
IviScope_FetchWaveform(ViSession vi,
    ViConstString channel,
    ViInt32 waveformSize,   // # of elements on input
    ViReal64 waveform[],    // actual data buffer
    ViInt32 *actualPoints,  // # of elements on output
    ViReal64 *initialX,
    ViReal64 *xIncrement);
```
Simplifying APIs with .NET TTypes

IVI-C signature

```c
IviDigitizer_FetchWaveformReal64(ViSession Vi,
    ViConstString ChannelName,
    ViInt64 WaveformArraySize,
    ViReal64 WaveformArray[],
    ViInt64* ActualPoints,
    ViInt64* FirstValidPoint,
    ViReal64* InitialXOffset,
    ViReal64* InitialXTimeSeconds,
    ViReal64* InitialXTimeFraction,
    ViReal64* XIncrement);
```

IVI.NET signature

```c
Channels[].Measurement.FetchWaveform(IWaveform<Double> waveform)
```
Standard IVI.NET Data Types

- IVI Foundation felt it would be useful to offer commonly used data types as part of the IVI.NET Shared Components
  - Increase consistency amongst IVI.NET drivers
  - Facilitate data interchange between drivers

- Standardized IWaveform and ISpectrum interfaces
  - Digitizers and scopes and RF spectrum analyzers all read waveforms
  - Function generators and RF signal generators source waveforms
  - Without a common definition of a “waveform”, client applications would need to write the tedious code to translate between each class’s notion of a waveform

- Time-based parameters can use PrecisionDateTime and PrecisionTimeSpan
  - No confusion about ms vs sec, absolute vs relative time, UTC time, etc
  - Precision adequate for IEEE 1588 devices

- Common trigger source data type
  - Useful in “stitching” together devices in triggered source-measure operations
How to deal with Events?

- **IVI-COM and IVI-C drivers almost never expose events**
  - Exposing something as commonly needed as an SRQ is tortuous
  - Requires special knowledge/programming by the driver developer
  - Requires special knowledge/programming by the client programmer

- **.NET supplies a standard mechanism for exposing events**
  - No special programming required by the driver developer or client programmer
  - Trivial code to subscribe/unsubscribe
  - Trivial code for driver developers to customize subscribe/unsubscribe semantics

- **Warnings can now be dealt with properly in IVI drivers by the use of events**
Error Handling in IVI.NET

• IVI-C drivers rely solely on return codes
  • Errors can easily be ignored by the client application
  • After getting the error code, a second function call is required to get the message
  • Special handling of warning codes required

• IVI-COM error code handling depends upon the client environment
  • Return codes in raw C++
  • Special exception classes in C++
  • COMException class thrown in .NET environments (based on interop)

• IVI.NET drivers always use exceptions
  • User can always see the full context of the error
  • Error content less dependent upon specific driver implementation
  • Natural mechanism
More Information

• For more information
  – IVI Website:  www.ivifoundation.org
  – IVI Web Forum:  forums.ivifoundation.org
  – IVI Driver Registry: http://www.ivifoundation.org/registered_drivers/driver_registry.aspx
    – Most vendors have documentation and drivers on their website
• For questions on IVI, go to the IVI Web Forum, open an account, and get answers from member companies.
  • IVI Web Forum:  forums.ivifoundation.org
Simplifying Test System Development with IVI.NET

Kirk Fertitta
Pacific MindWorks