User-Focused
IVI-COM Driver Development

Rengan Rajendran, Vektrex Electronic Systems, Inc., rrajendran@vektrex.com

Abstract - The IVI standards describe a rich, flexible architecture for instrument drivers. IVI's advanced features, such as the Component Object Model (COM) interface, offer unprecedented capability for instrument driver developers. With this capability comes a challenge – the challenge to produce drivers that meet the IVI standards while also providing excellent end-user utility. Though the IVI specifications are voluminous, many development decisions are left up to the individual driver developer. These developer choices can have a significant positive or negative impact on the utility of the resultant driver.

This paper describes best practices for developing user-focused IVI-COM drivers – drivers that present instruments in an intuitive way, accelerating access to measurement data. The paper will cover the following:

- **Overall driver API design in various situations, including situations in which existing plug&play drivers are being ported to IVI-COM**
- **Driver tuning for various client ADEs, including usage of wrappers**
- **Testing and validation strategies to assure conformance with IVI specifications**
- **Help files to educate and inform end-users**
- **Optimal driver integration with the IVI shared components**

1. INTRODUCTION

The IVI Foundation has written a robust specification that details the architectural requirements for IVI drivers. The specifications enforce both architecture and end-user needs but there are many areas where the driver developer has flexibility. This paper examines these areas and provides guidelines to ensure that drivers developed are user-focused. This paper focuses on IVI-COM drivers. However, the concepts can be applied to all IVI drivers. Throughout this paper, a sample IVI-COM driver that supports the IviScope instrument class will be used to illustrate the concepts. This driver is *VektrexScope*.

2. DRIVER DESIGN

The contents of the Inherent and class compliant interfaces are fixed by the IVI Foundation specifications. The only API that the driver developer has control over is the instrument specific API. Therefore, the first step in developing a user-focused IVI-COM driver is to design a comprehensive and intuitive instrument specific API.

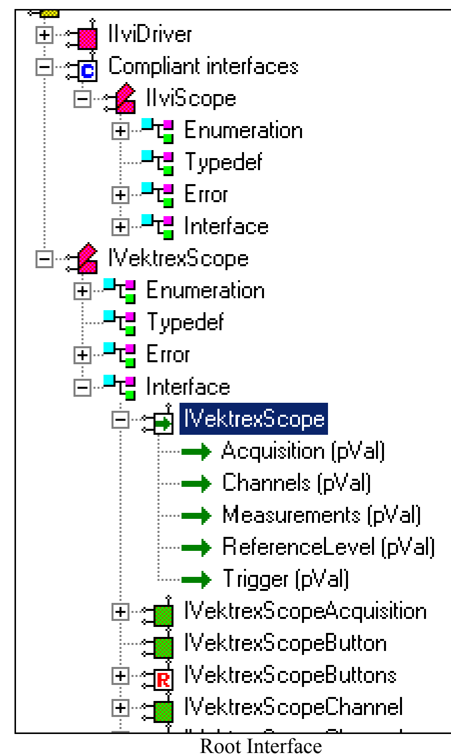
2.1 Instrument Specific Interface API Design

The Instrument Specific Interfaces should contain methods and properties to access the most commonly used functionalities of the instrument. Where offered by the instrument, IVI compliant capabilities should be mirrored in the instrument specific interfaces. In fact, inside the driver, the IVI-compliant function will often call the instrument specific version.

2.1.1 Identifying and Naming Interfaces

Before any properties or methods are defined, the instrument specific interfaces require definition. Guidelines for defining interfaces are:

- The root interface is always the name of the instrument or instrument family, e.g., *IVektrexScope*.



Root Interface

The root interface usually only contains interface reference properties. When the driver is implemented, the inherent methods *Initialize* and *Close* and the *get_Initialized* property

are included in the root interface through inheritance from IIviDriver. Use the IVI class specification as a guide. Take advantage of interface designs already developed on a broad scale.

- Break down the functionality of the device as seen from the user's perspective. The client application developer should be able to make intuitive choices as to which interface to use. The operation through the driver should mimic the operation through the instrument's front panel.

- Use interface names already used in the IVI class specification. If the specific interface closely matches the compliant interface, use the same name. The driver prefix to the name will ensure uniqueness.

- Look at the instrument's documentation. The instrument designers needed to solve the same "interface" problems when they designed the *user* interface. The structure of the instrument documentation may yield clues to duplicating that same user interface.

- Follow conventions already in place. Revisions of a driver should maintain a certain level of consistency not only for readability, but also so client applications will continue to run without failure.

- Break large interfaces into smaller interfaces. Group like functionality interfaces into smaller groups, to avoid creating a super, unmanageable interface. For example, a *Waveform* interface may be broken down into a *StandardWaveform* interface and an *ArbitraryWaveform* interface.

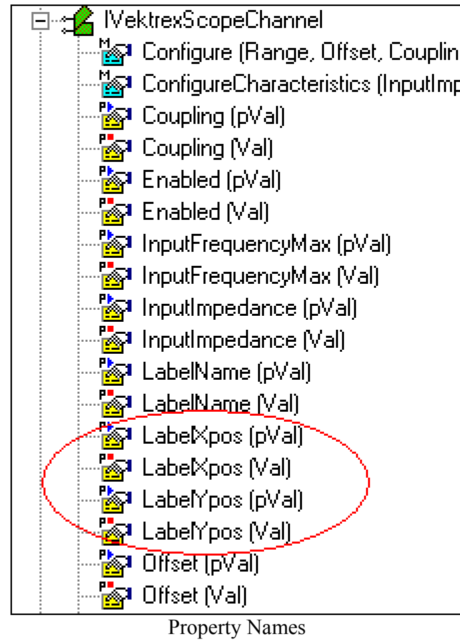
- Limit the number of interface reference properties in an individual interface to between 6 and 12. For LabVIEW support, this number *must* be less than 30.

By convention, interface names begin with the capital letter "I", so the above interface names would be *IStandardWaveform* and *IArbitraryWaveform*.

Interface names are very important. Choose intuitive names.

2.1.2 Naming Properties

Property names, for the most part, take their names from the state or information in the instrument or driver that they represent. Within a given interface, a property should be named for what it represents. A property named *State* is ambiguous on its own, but in the context of the interface *Calculation*, no meaning is lost. There is no need to make the property name redundant with the interface name (e.g., a property name of *CalculationState* is not needed). Spell out the words in English; avoid abbreviations unless the abbreviation is very well known, such as AM, FM, dB, Log or Hz. Be consistent with word order. If a word appears in multiple properties, put it first before any modifying words. For example in the *Channel* interface of *VektrexScope* there are properties called *LabelName, LabelXpos* and *LabelYpos*.



Property Names

A word naturally associated with the attribute's units should appear first. For example, a value with units of Hertz should have a name that starts with frequency. A useful side effect of these rules becomes apparent when attributes are listed alphabetically. Related attribute names and values are listed together.

Choose intuitive names for properties.

2.1.3 Naming Methods

The conventions for naming properties apply to methods except the method name inherently describes an action to perform (e.g., *Configure, Measure* or *Read)*. The first word should be a verb or at least imply performing an action. Spell out the words in English using mixed case; avoid abbreviations. Assume a naming convention based on the context of the interface in which the method resides.

Choose intuitive names for methods.

2.1.4 Methods Represent Clusters of Functionality

There are groups of instrument properties that the developer logically wants to group together. Often, a set of instrument parameters or modes will be set "simultaneously" by the end user's client application. Examples of such a grouping are:

- Function, Range and Resolution

- TriggerSource and TriggerDelay

- StartFrequency, StopFrequency, SweepTime, and Spacing

Grouping multiple instrument properties with related functionality is accomplished by creating methods that set a group of properties. Grouping of related properties with a method simplifies the end user's client application code. A single method accomplishes the task of multiple properties.

Methods that set multiple properties need to account for coupling. Instrument properties should be set in the correct order to avoid unwanted errors or unpredictable results. A single method that sets all the instrument properties according to the hierarchical coupling rules takes the onus off the end user to reproduce the correct precedence.

Abstract common sets of properties as methods.

2.2 Leveraging Existing VXI PnP Drivers

IVI-COM drivers provide a better end-user experience than VXI PnP drivers. However, it is not always feasible to *rewrite* every PnP driver as an IVI-COM driver. An option that is available is to wrap existing, robust, field-tested PnP drivers. The wrapped driver can take advantage of the existing driver code that communicates with the instrument while presenting an IVI-COM interface to client environments.

Take advantage of existing driver code if at all possible.

3. DRIVING TUNING FOR CLIENT ADES

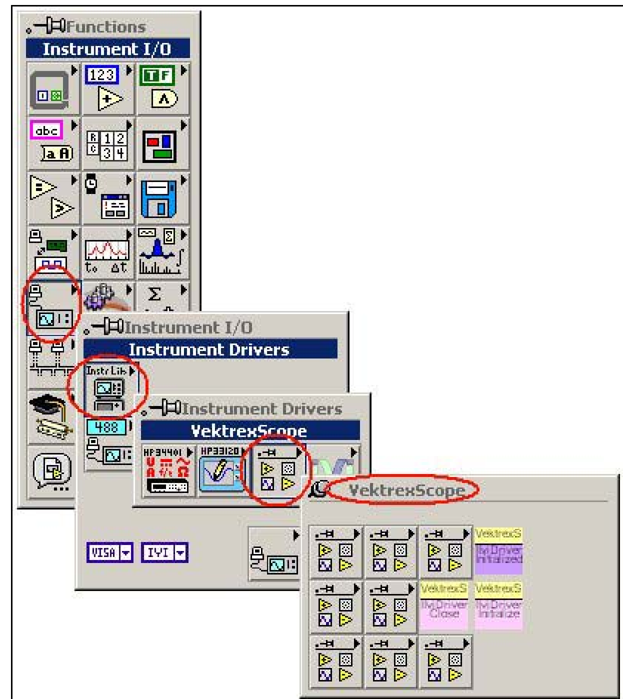
IVI-COM drivers work in all client application development environments that support custom COM interfaces. However, the end-user experience is better in some environments than others. Visual Basic and Visual C++ provide natural and robust environments for IVI-COM drivers. With a little bit of work, LabVIEW and .NET environments also become natural and robust client environments.

3.1 LabVIEW Wrappers

LabVIEW Wrappers provide the client application developer a way to directly access an IVI-COM driver from the National Instruments LabVIEW development environment. Specifically, wrappers allow IVI-COM instrument drivers to be placed into the **Functions | Instrument I/O | Instrument Drivers** palette, visible from the diagram-view in National Instruments LabVIEW. Each method and property of the instrument's IVI-COM driver is available via its own VI.

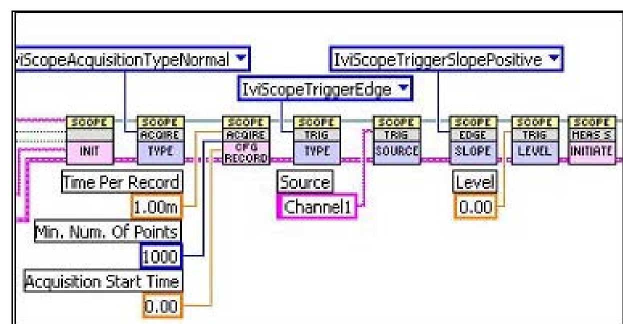
Wrapper VIs greatly simplify the IVI-COM driver experience for LabVIEW users. Instead of wiring numerous property nodes together to access a driver object, users simply select VIs corresponding to driver methods and properties. The VIs are organized into cascading palettes that follow the driver interface hierarchy. Clicking on a VI icon automatically brings up the Windows driver help file for in-depth information about a particular method or property.

For instance, the following figure displays the Vektrex Oscilloscope IVI-COM driver.



Accessing Wrappers in LabVIEW

With LabVIEW wrappers, accessing IVI-COM drivers becomes as easy as using any other driver as shown in the following example:



LabVIEW Wrapper Application Example

IVI-COM driver providers should provide LabVIEW wrappers for their drivers.

3.2 .NET Primary Interop Assemblies (PIAs)

.NET clients access IVI-COM drivers through wrappers called interop assemblies.

To use an IVI-COM driver in the Visual Studio .NET programming environment, the application developer adds a reference to the driver. If a primary interop assembly, PIA, is not registered for the driver, the development environment creates an interop assembly for the project. If the driver has a registered PIA, the development environment uses that PIA.

Driver developers need to ensure that .NET programmers are using the same PIAs and not inadvertently introducing type incompatibilities into their programs. A fundamental rule of .NET is that the assembly that defines the type, scopes the type. A type with the same name and definition that is defined in two different assemblies is considered to be two different types by the .NET runtime. By following the rules for generating a single PIA for an IVI-COM driver, a driver developer ensures that type incompatibilities are not introduced.

IVI-COM driver providers should provide Primary Interop Assemblies for their drivers.

4. TESTING AND VALIDATION STRATEGIES

Driver users want to be sure that drivers are conformant to IVI Specifications and are easy to use. The following is a checklist of areas that need to be validated. This checklist was generated from years of experience working with IVI drivers and should be used as a reference along with compliance requirements from the IVI Foundation.

- API testing
    - Verify that instrument specific interfaces, properties, and methods follow the IVI guidelines for naming.
    - Exercise every method and property and verify that they return valid values
    - Verify that class extensions that are not implemented return valid error codes.

- Verify that the driver works in simulation

- Verify that all interfaces are accessible through interface reference properties starting from the root of the driver

- Verify locking

- Verify that methods and properties are not accessible (i.e., return an error) before the driver is initialized except for the limited number of inherent methods and properties that are defined to be always accessible

- Test range checking, coercion, and state caching

- Verify that the driver handles virtual name mapping properly

- Verify the "semantic" behavior of the driver

- Verify that the context-sensitive help file works properly from multiple client environments

- Verify that the driver is usable from all applicable client development environments. This includes Visual Basic, Visual C++, LabVIEW, .NET clients, and LabWindows.

- Installation testing – verify the following:
    - All the files are installed including the driver dll and help file
    - That the Configuration Store is populated correctly
    - That the registry is properly updated including settings for the help file
    - That a compliance file is installed

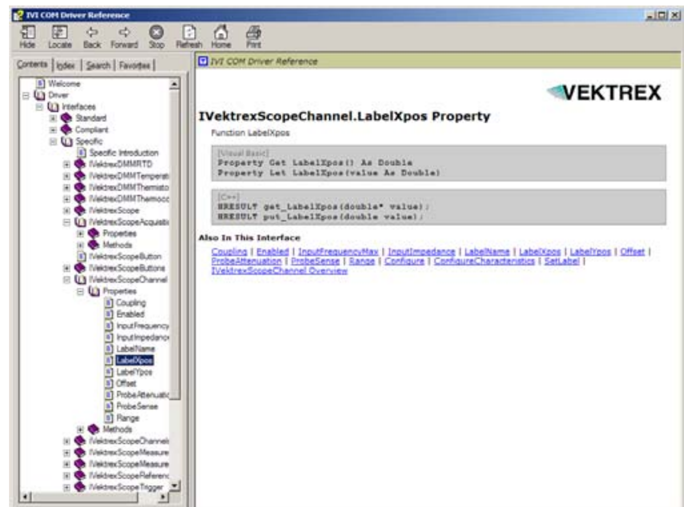
- Un-installation testing – verify the following:
    - That the proper entries are removed from the Configuration Store
    - That the registry is restored
    - That all the installed files are removed

Use a comprehensive checklist to validate your drivers.

5. HELP FILES

The IVI Specifications require that help files be deployed with drivers. However, there are no guidelines for these instrument specific help files that are a very important aspect of the client end-user experience. The following guidelines should be followed:

- Help files should be context-sensitive.
    - It should be possible to call up the help file in-context from a client application development environment. The following screen shot shows the help that is brought up when F1 is pressed on the LabelXpos property in Visual Basic.



VektrexScope Help File

- Help files should follow a consistent style. Each driver vendor should define a "template" for help files and follow the template rigorously.

- Help files should provide usage examples. Examples of using a driver from Visual C++ and Visual Basic are shown in the above example.

- Help files should provide an overview for the driver. The overview should include whether the driver supports a class. If a class is supported, the supported extension groups should also be listed.

- Help files **may** provide help for the class. The IVI Foundation provides help files for each of the IVI defined classes. When a user presses F1 (in Visual Basic) on a class method or property, the IVI Foundation help file will be launched. The information provided in this help file is generic class help. If the specific vendor class implementation has limitations or any caveats, these should be captured in the class compliant help in the Instrument Specific help file.

Provide context-sensitive, robust, consistent help files.

6. INTEGRATION WITH IVI SHARED COMPONENTS

The IVI Foundation has designed and documented a robust set of requirements for driver installations. The requirements address the many aspects of driver installations including:

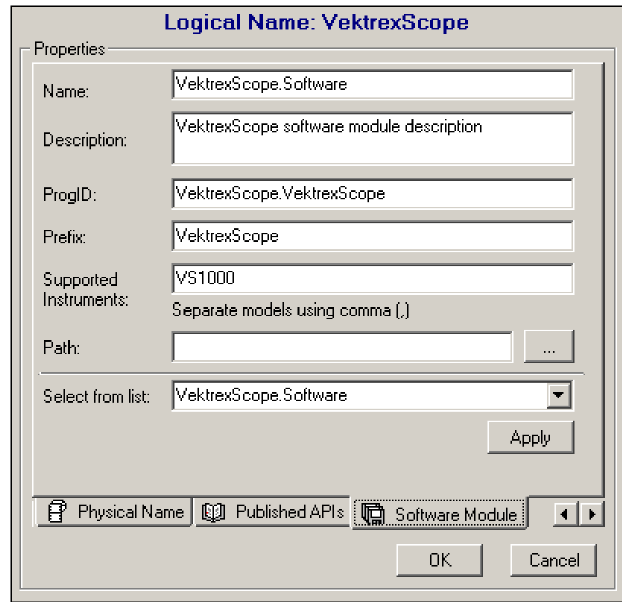
- Checking for the presence of IVI Shared Components and installing them if they are not present or if they are of the wrong version

- Installation directories including driver and documentation locations

- Registry entries

- Entries in the IVI defined Configuration Store parameter configuration file

It is in the Configuration Store entries where driver installs can be more user-focused. The IVI Foundation defines the *minimal* set of entries that need to be added to the Configuration Store. The only entry that is required is the software module entry. The software module entry describes the driver dll that has been installed as shown below.



Software Module Information

With just the software module entry in the Configuration Store, the drivers are not usable "out-of-the-box" if the client application wants to take advantage of the Configuration Store data. This is true even in simulation mode. The driver can be used explicitly without accessing the Configuration Store. For example if an explicit reference is added in Visual Basic to the VektrexScope driver, the following can be used to instantiate and initialize the driver:

```
Dim myscope As New VektrexScope

myscope.Initialize "GPIB0::20::INSTR", True, True, ""
myscope.Channels.Item("Channel1").LabelXpos = 1.02
```

The following additional entries need to be minimally added to the Configuration Store to use the driver implicitly:

- Driver Session

- Hardware Asset

Ideally, the following additional information needs to be set up:

- Logical name

- Virtual name mapping (if applicable)

If just the software module entry is added to the Configuration Store, the user will need to manually go through the following steps:

- Create a driver session

- Find (this is not necessarily a trivial task) the software module that in the Configuration Store and add a reference to it.

- Add a hardware asset and add a reference from the driver session

- Add virtual names and associate them with the driver session

It is much simpler to have the driver installation add some default values for Driver Session, Hardware Asset, Logical Name, and Virtual Names and have the user modify these values if needed. The following set of diagrams show the additional fields that have been added.

Logical Name: VektrexScope
Properties
Name: VektrexScope
Description: VektrexScope demonstration driver
Apply
Logical Name | Session | H/W Asset | Virtual Name
OK Cancel

Logical Name Information

Logical Name: VektrexScope
Properties
Name: VektrexScope.DriverSession
Description: VektrexScope demonstration driver session
Driver Setup:
Software Module: VektrexScope.Software
Check the supported propties
Cache ☑ Interchange ☐ Range Check ☐ Simulate ☐
Query Instrument Status ☑ Coercions ☐
Select from list: VektrexScope.DriverSession
Apply
Logical Name | Session | H/W Asset | Virtual Nam
OK Cancel

Driver Session Information

Logical Name: VektrexScope
Properties
Name: VektrexScope.Hardware
Description: VektrexScope demonstration driver hardware asset
I/O Resource: Descriptor GPIB0::20::INSTR
Select from list: VektrexScope.Hardware
Apply
Logical Name | Session | H/W Asset | Virtual Name
OK Cancel

Hardware Asset Information

Logical Name: VektrexScope
Properties

Virtual Name	Map To
Ch1	Channel1
Ch2	Channel2
Me1	Measurement1
Me2	Measurement2

Add Remove Modify
Range

Name	Min	Max	Index

Apply
Logical Name | Session | H/W Asset | Virtual Name
OK Cancel

Virtual Name Information

If the above entries are added to the Configuration Store, it is possible to access the driver in the following manner:

```
Dim myscope As New VektrexScope

myscope.Initialize "VektrexScope", True, True, ""
myscope.Channels.Item("Ch1").LabelXpos = 1.02
```

With these additional entries it is also possible to use the IVI Session Factory to access the driver.

```
Dim mysession As New IviSessionFactory
Dim myscope As IIviScope

myscope = mysession.CreateDriver("VektrexScope")
myscope.Initialize "VektrexScope", True, True, ""
```

Simplify the usage of IVI Shared Components for end-users.

7. CONCLUSION

The IVI Foundation has written robust specifications that detail architectural requirements that also address end-user needs. However, there are areas that are left open to interpretation. These areas include driver design, client environment support, validation drivers, help files, and deployment. This paper provides some guidelines and best practices for these areas resulting in drivers that are more end-user focused.

8. ABOUT VEKTREX

Incorporated in 1986, Vektrex is a software and systems integration company supporting the test and measurement industry. The company is focused on advancing measurement technology and migrating measurements from hardware; test instruments and test tubes, into software and the PC. Vektrex has established key expertise with test instruments, instrument drivers, and software based measurements and provides enabling products and services that help clients migrate to software-based measurements. Products include patent pending Masquerade technology enabling the rapid development of Composite Replacement Instruments (CRI); form, fit, function replacements for obsolete instrumentation in legacy ATE. Vektrex also produces software products focused on drivers and reusable software components including VIVID, the software development toolkit that produces IVI compliant COM based drivers. Vektrex' clients span advanced technology industries including defense, biotechnology, and semi-conductor. See Vektrex at www.vektrex.com.