IVI Fundamentals

Outline

	0:10	IVI Overview (the Foundation and Driver overview)
	0:45	Using IVI – an example application
	0:20	IVI Repeated capabilities
	0:25	IVI Coding Patterns and Features
	0:30	Driver Advanced Topics
		Conclusion

IVI Overview

Purpose: Summarize the work of the consortium
Introduce the technical material with big picture

Topics:

- The Consortium
- The Driver Specifications
- Benefits and Features of the Drivers

Time: 00:15

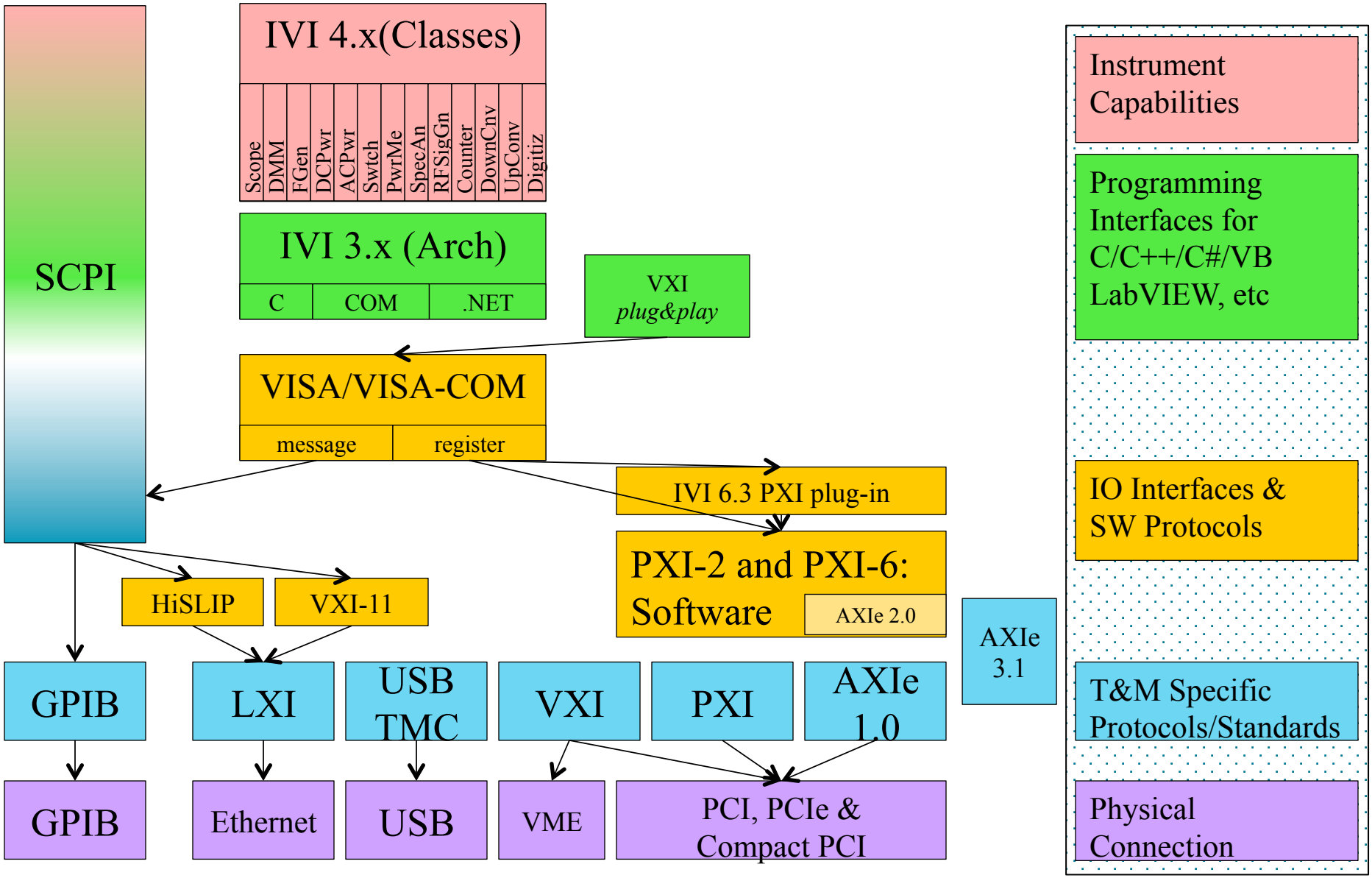
What is IVI?

The primary purpose of the Consortium is to:

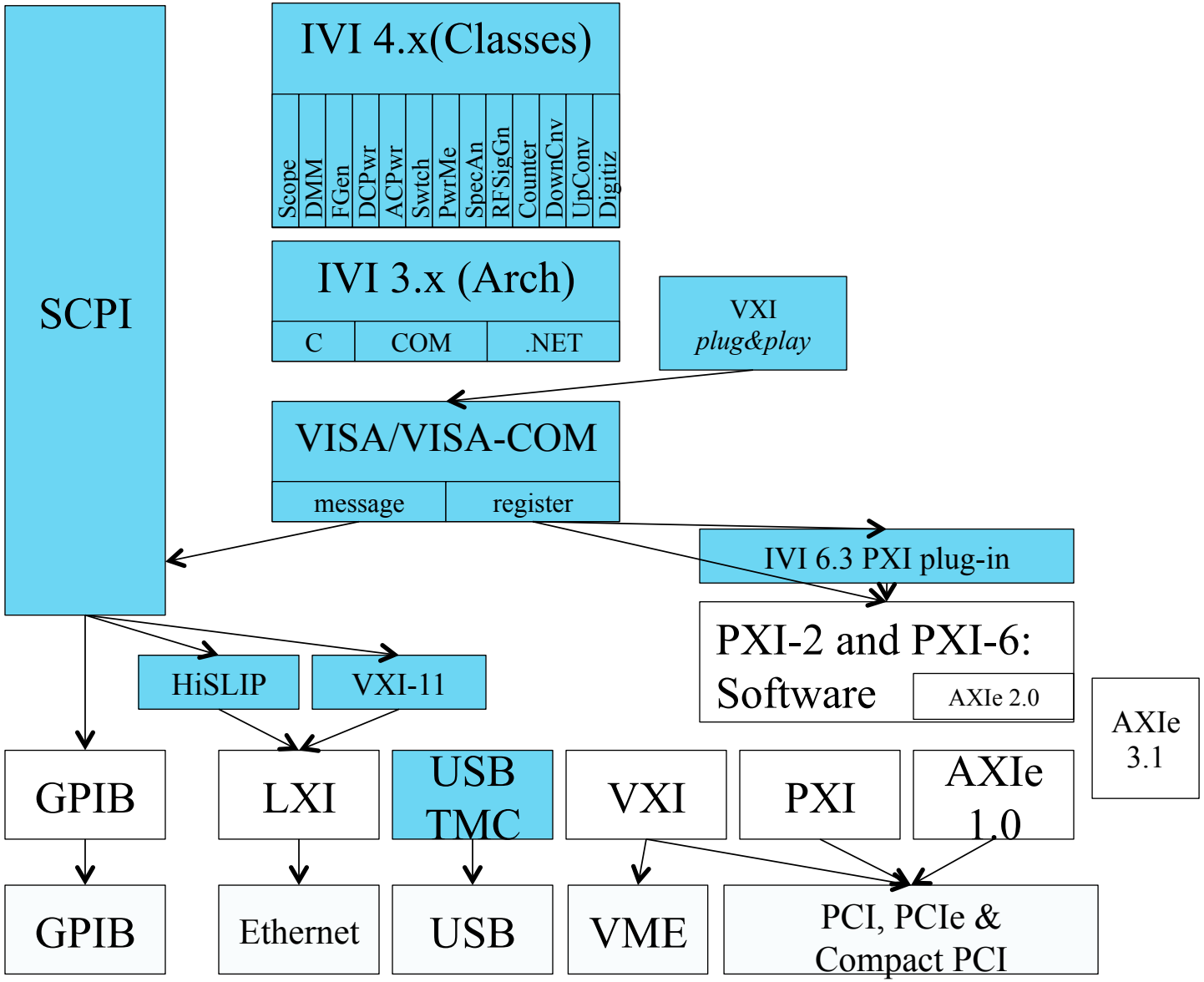
- *Promote the development and adoption of standard specifications for programming test instrument*
- *Focus on the needs of the people that use and develop test systems who must take off-the-shelf instrument drivers and build and maintain high-performance test systems*
- *Build on existing industry standards to deliver specifications that simplify interchanging instruments and provide for better performing and more easily maintainable programs*

From IVI Foundation *By-laws*

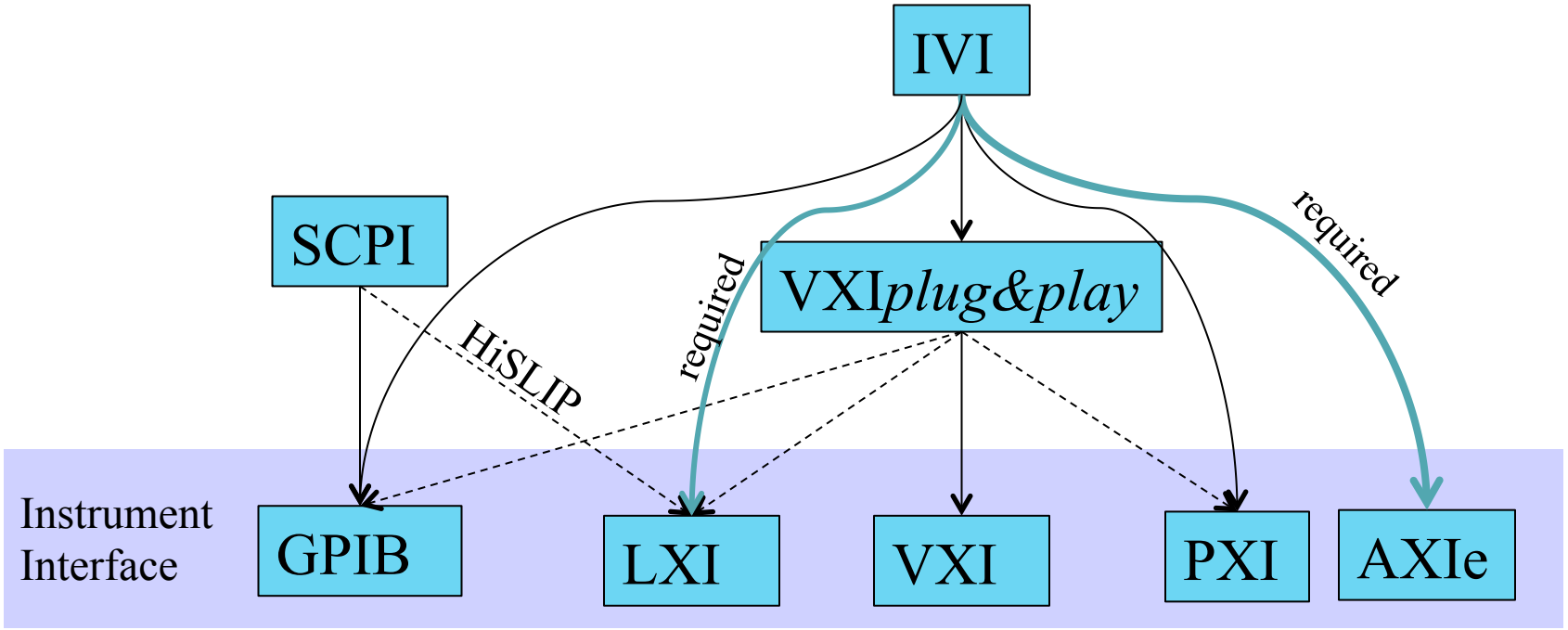
IVI fit with other specs



IVI fit with other specs



IVI Fit with Other Specs



- SCPI provided necessary standards based on GPIB needs
  - Command strings natural match to GPIB
  - HiSLIP allows connection to LXI
- VXI*plug&play* added drivers necessary for VXI
  - Used with other I/O to provide necessary driver
- IVI enhances VXI*plug&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

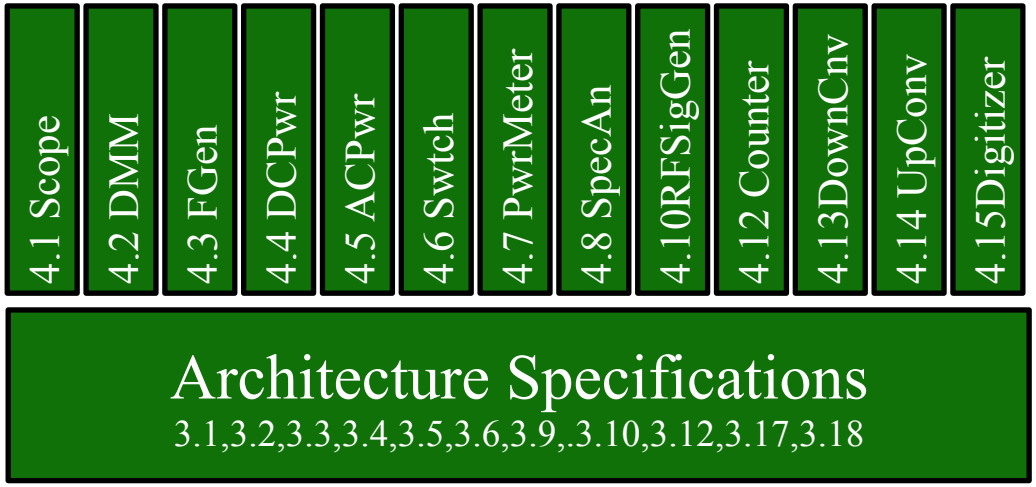
- Programming with IVI-C

```
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 ….
- Consistent API
  - Common organization, data types, naming

Class Compliant

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

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? – 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

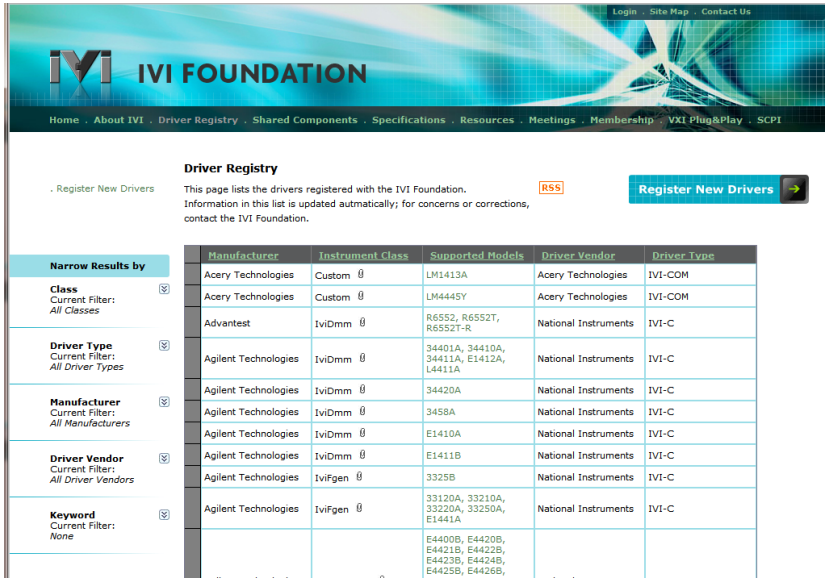
IVI-2014

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



End of IVI Intro

Using an IVI Driver

Purpose: Show how to get started with IVI drivers

Topics:

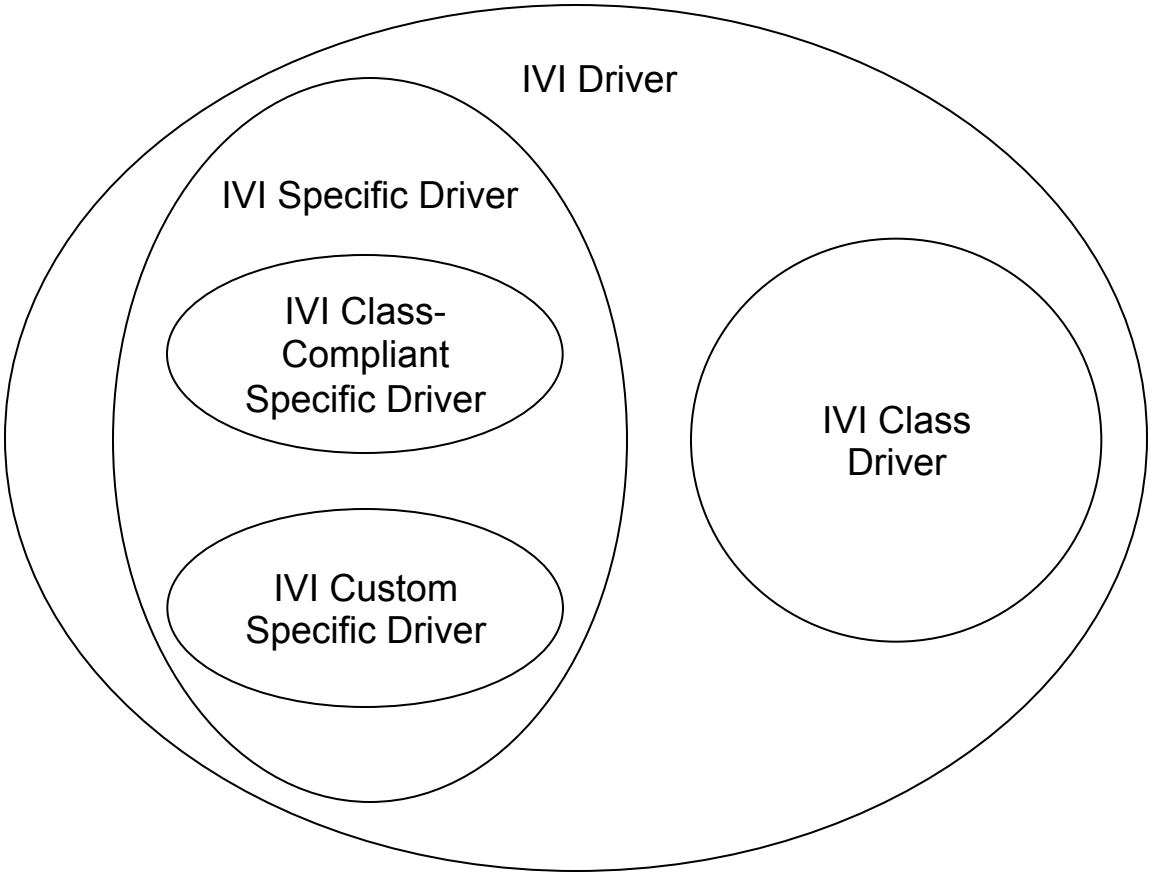
- Basic use (Open/Close/Function/Attribute)
- Types of IVI drivers
- Repeated capabilities

Time: 00:45

Formal IVI Driver Terminology

- IVI Driver
  - Implements IVI-3.2: Inherent Capabilities Specification
  - Complies with all of the architecture specifications
  - May or may not comply with an instrument class
  - Is either an IVI specific driver or an IVI class driver
- IVI Specific Driver
  - Written for a particular instrument
- IVI Class-compliant Specific Driver
  - IVI specific driver that complies with one (or more) of the IVI-defined class specifications
  - Used when hardware independence is desired
- IVI Custom Specific Driver
  - IVI specific driver that is not compliant with one of the IVI-defined class specifications
  - Not interchangeable
- IVI Class Driver
  - IVI driver needed only for interchangeability in IVI-C environments
  - Class may be IVI-defined or customer-define

Types of IVI Drivers



IVI Driver Terminology in Practice

- Formal terminology is confusing
  - Overly generic because of need to describe IVI-C, IVI-COM and IVI.NET drivers
  - Much simpler in practice
- For IVI-COM and .NET:
  - There is no class-driver (more on this later), so the distinction between “class driver” and “specific driver” is not applicable
  - Only one software component (the specific driver or simply, the “driver”)
    - Implements instrument-specific interfaces and (optionally) class-compliant interfaces
- For IVI-C:
  - Class driver required for interchangeability
    - Delegates function calls to the specific driver
    - National Instruments is the only supplier (free download)
  - Vendors provide a “specific” driver as a separate component

Three API Standards in IVI

- IVI class and architecture specifications layout rules for all APIs
  - IVI-COM
  - IVI-C
  - IVI.NET
- To choose which one you need, consider:
  - Development environment (C, .NET, LabVIEW, MATLAB, …)
- IVI specs also define and specify “driver wrappers”
  - Specifies how to support multiple types of drivers (C, COM, .NET)
  - IVI-COM on top of IVI-C (unusual)
  - IVI-C on top of IVI-COM (common)

Functions and Attributes

- IVI uses the generic terms functions, attributes to refer to the elements of the API exported by an IVI driver
- Functions
  - Refers to standard COM methods in IVI-COM
  - Refers to C entry point functions in IVI-C
- Attributes
  - Properties in IVI-COM and IVI.NET
    - [propput] and [propget] used in COM IDL definitions
  - IVI-C uses attribute access functions
    - Functions for Get and Set of various data types

```
Ag34401_GetAttributeViReal64(vi, "", Ag34401_ATTR_RANGE, &range)
```

Attribute Values

- Attribute Values
  - Numeric or discrete
  - Discrete values represented by enums in IVI-COM
    - Example: IviScopeTriggerTypeEnum, with defined enum values
    - Enum defined in class-compliant type library provided by IVI
  - Discrete values represented by #defined values (macros) in IVI-C
    - Example: IVISCOPE_VAL_EDGE_TRIGGER
    - Values defined in header file (iviscope.h) supplied by specific or class driver

Demonstration: C# Hello World

- This demonstration shows
  - Open/Close a driver
  - Set property
  - Call a method
- Using IVI-C C++ Demo

Demonstration: C Hello World

- This demonstration shows
  - Creating a C project calling IVI Library
  - Opening a driver in C
  - Calling some functions to get a result
  - Setting and getting an attribute
- Using IVI-C CVI Demo

Simple IVI-COM Driver Usage in Visual Basic

```
‘ Note: Direct reference to Agilent54600 makes
‘ this code not interchangeable

Dim driver as New Agilent54600
Dim scope as IIviScope
Set scope = driver

scope.Initialize “GPIB::10”, False, False, “”

scope.Trigger.Level = 3.4                    ‘setting a numeric property
scope.Trigger.Type = IviScopeTriggerEdge     ‘setting an enum property

scope.Measurements.Initiate                        ‘calling a method

scope.Close
```

Simple IVI-C Driver Usage

```
// Statically link with ag54600.lib
// NOTE: Many changes required to make interchangeable
#include “ag54600.h”

int _tmain(int argc, _TCHAR* argv[])
{
   ViSession vi;
   ViStatus viStatus = ag54600_init(“GPIB::10”, VI_FALSE, VI_FALSE, &vi);

   viStatus = ag54600_SetAttributeViReal64(vi, “”,
                                           AG54600_ATTR_TRIGGER_LEVEL, 3.2);

   viStatus = ag54600_SetAttributeViInt32(vi, “”,
                                          AG54600_ATTR_TRIGGER_TYPE,
                                          AG54600_VAL_EDGE_TRIGGER);
   viStatus = ag54600_InitiateAcquisition();
   viStatus = ag54600_close();
}
```

Simple IVI-C Driver Usage

```
// Statically link with ag54600.lib
// NOTE: Many changes required to make interchangeable
#include “ag54600.h”

int _tmain(int argc, _TCHAR* argv[])
{
   viSession vi;
   viStatus viStatus = ag54600_init(“GPIB::10”, VI_FALSE, VI_FALSE, &vi);

   viStatus = ag54600_SetAttributeViReal64(vi, “”,
                                           AG54600_ATTR_TRIGGER_LEVEL,
                                           3.4)
   viStatus = ag54600_ConfigureTrigger (vi,
                                        AG54600_VAL_EDGE_TRIGGER,
                                        200e-9)


   viStatus = ag54600_InitiateAcquisition();
   viStatus = ag54600_close();
}
```

IVI-C Attribute Accessors

```
// Corresponding group of setters also exist
Prefix_GetAttributeViInt32 (ViSession vi, ViConstString RepCapIdentifier,
                            ViAttr AttributeID,
                            ViInt32 *AttributeValue);

Prefix_GetAttributeViReal64 (ViSession Vi, ViConstString RepCapIdentifier,
                             ViAttr AttributeID,
                             ViReal64 *AttributeValue);

Prefix_GetAttributeViBoolean (ViSession Vi, ViConstString RepCapIdentifier,
                              ViAttr AttributeID,
                              ViBoolean *AttributeValue);

Prefix_GetAttributeViSession (ViSession Vi, ViConstString RepCapIdentifier,
                              ViAttr AttributeID,
                              ViSession *AttributeValue);

Prefix_GetAttributeViString (ViSession Vi, ViConstString RepCapIdentifier,
                             ViAttr AttributeID,
                             ViInt32 AttributeValueBufferSize,
                             ViChar AttributeValue[]);
```

IVI Capability Groups

- Inherent IVI Capabilities
  - Functionality all IVI drivers must implement
  - Defined in IVI-3.2: Inherent Capabilities Specification
- Instrument-Specific Capabilities
  - Instrument functions independent of the instrument class specification
  - Defined by the driver developer
- Base Class Capabilities
  - Functionality common across all instruments of a particular class
  - Required to implement for class-compliant drivers
  - Defined in the relevant instrument class specification
- Class Extension Capabilities
  - More specialized features of an instrument class
  - Not required to be class-compliant
  - Defined in the relevant instrument class specification

Custom Instrument Specific driver

Class Compliant Specific driver

IVI Inherent Capabilities – IVI-COM

COM Interface Hierarchy	Type
DriverOperation	
Cache	P
ClearInterchangeWarnings	M
DriverSetup	P
GetNextCoercionRecord	M
GetNextInterchangeWarning	M
InterchangeCheck	P
InvalidateAllAttributes	M
LogicalName	P
QueryInstrumentStatus	P
RangeCheck	P
RecordCoercions	P
ResetInterchangeCheck	M
IoResourceDescriptor	P
Simulate	P

COM Interface Hierarchy	Type
Identity	
Description	P
Identifier	P
Revision	P
Vendor	P
InstrumentManufacturer	P
InstrumentModel	P
InstrumentFirmwareRev…	P
SpecificationMajorVersion	P
SpecificationMinorVersion	P
SupportedInstrumentModels	P
GroupCapabilities	P

COM Interface Hierarchy	Type
Close	M
Initialize	M
Initialized	P
Utility	
Disable	M
ErrorQuery	M
LockObject	M
Reset	M
ResetWithDefaults	M
SelfTest	M
UnlockObject	M

M = Method
P = Property

Example Class Capabilities

Group Name	Description
IviScopeBase	**Base Capabilities: Includes the capability to acquire waveforms using edge triggering.**
IviScopeInterpolation	**IviScope with the ability to configure the oscilloscope to interpolate missing points in a waveform.**
IviScopeTVTrigger	**IviScope with the ability to trigger on standard television signals.**
IviScopeRuntTrigger	**IviScope with the ability to trigger on runts.**
IviScopeGlitchTrigger	**IviScope with the ability to trigger on glitches.**
IviScopeWidthTrigger	**IviScope with the ability to trigger on a variety of conditions regarding pulse widths.**
IviScopeAcLineTrigger	**IviScope with the ability to trigger on zero crossings of a network supply voltage.**
IviScopeWaveformMeas	**IviScope with the ability to calculate waveform measurements, such as rise time or frequency.**
IviScopeMinMaxWaveform	**IviScope with the ability to acquire a min and max waveforms that correspond to the same time range.**
IviScopeProbeAutoSense	**IviScope with the ability to automatically sense the probe attenuation of an attached probe.**
IviScopeContinuousAcquisition	**IviScope with the ability to continuously acquire data from the input and display it on the screen.**
IviScopeAverageAcquisition	**IviScope with the ability to create a waveform that is the average of multiple waveform acquisitions.**
IviScopeSampleMode	**IviScope with the ability to return the actual sample mode.**
IviScopeTriggerModifier	**IviScope with the ability to modify the behavior of the triggering subsystem in the absence of a trigger.**
IviScopeAutoSetup	**IviScope with the ability to perform automatic configuration.**

End of Using an IVI Driver

IVI Repeated Capabilities

Purpose: Cover IVI Repeated Capabilities

Topics: Describe what IVI Repeated Capabilities are
Discuss 3 ways that they are implemented
Use with the IVI Config Store
Show their use in some examples

Time: 00:30

Repeated Capabilities

- Many instruments contain multiple instances of the same type of functionality – IVI terms these *repeated capabilities*
  - Example: Channels in an oscilloscope
  - Example: Traces or markers in a spectrum analyzer
- An instrument may have multiple sets of repeated capabilities
  - Example: A scope with channels and traces
  - Example: A device with analog channels and digital channels
- Repeated capabilities can be nested
  - Example: Traces within displays
- IVI specifies 3 ways drivers can implement repeated capabilities
- Classes partially specify repeated capabilities
  - Defines which functions and attributes apply to repeated capabilities

Repeated Capability Concepts

- Repeated capability name
  - Unique designator for a specific repeated capability in an instrument class
  - Example: IviScope spec defines “Channel” as a repeated capability name
  - Example: IviSpecAn spec defines “Trace” as a repeated capability name
- Repeated capability identifier
  - Unique designator for an instance of a particular repeated capability
  - Examples: “CH1”, “CH2” represent different instances of the “Channel” repeated capability
  - Two types exist to facilitate interchangeability: physical and virtual repeated capability identifiers
- Physical repeated capability identifier
  - Defined by specific driver
  - Placed in IVI Configuration Store by specific driver installer
- Virtual repeated capability identifier
  - Defined by end-user
  - End user maps virtual name to physical name in IVI Configuration Store
  - Required for interchangeable code

Repeated Capability Concepts

Repeated Capability Name: Channels

Defined by driver or class

Physical Repeated Capability Identifier: Chan1 Chan2 Chan3

Virtual Repeated Capability Identifier: Antenna PowerAmp Rotor

Defined by application

3 Ways to Expose Repeated Capabilities

- Parameter-style (pass element to every call)
  - Most common technique in IVI-C drivers
  - First parameter to each applicable function is a repeated capability identifier
  - Must include even if repeated capabilities are not applicable for instrument
    - Can pass in VI_NULL or an empty string if specific instrument has only one channel
- Selector-style (specify the element with a mode switch)
  - Special SetActive function used to set the active repeated capability identifier
    - All subsequent function/attribute calls use active repcap identifier
  - Useful if repcap identifier is complex and used repeatedly in a sequence of calls
- Collection-style (specify the element as a member of a collection)
  - Most common technique in IVI-COM drivers
  - Much simpler than other repcap styles when nesting is involved
  - Works a lot like standard COM collections
    - But w/o the nice VB for-each syntax

3 Ways to Expose Repeated Capabilities

- Parameter-style (pass element to every call)

```
AgM950x_FanTraySpeed(vi, "Tray1", &speed);
```

- Selector-style (specify the element with a mode switch)

```
AgM950x_FanTraySpeedSelect(vi, "Tray1");
AgM950x_FanTraySpeed(vi, &speed);
```

- Collection-style (element indexes into a collection)
  - Available (and preferred) with IVI-COM and IVI .NET (and preferred)

```
Int32 speed = myChassis.FanTray["Tray1"].FanTraySpeed
```

Repeated Capability Attributes and Functions

Technique	Attributes	Functions
Parameter	**<Capability> Count** **<Capability> Name (COM only)**	**Get <Capability> Name (C only)**
Selector	**<Capability> Count** **Active <Capability>** **<Capability> Name (COM only)**	**Get <Capability> Name (C only)** **SetActive <Capability>**
Collection*	**<Capability>s.Item** **<Capability>s.Count** **<Capability>s.Name****	***Not supported***

* IVI-COM collection attributes are placed in a collection interface with a name ending in <Capability> followed by an ‘s’.

** IVI-COM collections are 1-based

“Trace” Repeated Capability Example

- Class specification defines a “Trace” repeated capability

Technique	Attributes	Functions
Parameter	**TraceCount** **TraceName (COM only)**	**GetTraceName (C only)**
Selector	**TraceCount** **ActiveTrace** **TraceName (COM only)**	**GetTraceName (C only)** **SetActiveTrace**
Collection*	**Traces.Item** **Traces.Count** **Traces.Name****	***Not supported***

Parameter-Style Repeated Capabilities

```
‘ Repcap identifier must be passed to each method or property

Dim specan as New AgilentPSA

specan.Bandwidth.Item(“Trace1”) = 4E6
specan.Frequency.Item(“Trace1”) = 3E9
specan.Span.Item(“Trace1”) = 2E10
```

Selector-Style Repeated Capabilities

```
‘ Repcap identifier only specified once => convenient for complex identifiers

Dim specan as New AgilentPSA

specan.ActiveTrace = “Trace1,Trace2,Trace3”

specan.Bandwidth = 4E6
specan.Frequency = 3E9
specan.Span = 2E10
```

Repeated Capabilities Using Collections

- For each syntax not supported for IVI-COM collections
  - IVI-COM collections are not “real” COM collections
  - COM collections require IDispatch and IVI-COM interfaces are intentionally not IDispatch-based

```
Dim specan as New AgilentPSA
Dim trace as IAgilentPSATrace

Set trace = specan.Traces.Item(“Trace1”)

trace.Bandwidth = 4E6
trace.Frequency = 3E9
trace.Span = 2E10
```

Repeated Capabilities and IVI-C Attributes

- All IVI-C attribute accessors accept a repcap identifier as a parameter
  - Can pass VI_NULL or empty string if repeated capabilities do not apply to the attribute being read/written

```
agpsa_SetAttributeViReal64 (ViSession Vi, ViConstString RepCapIdentifier,
                            ViAttr AttributeID,
                            ViReal64 AttributeValue);
```

```
ViSession vi;
ViStatus viStatus = agpsa_init("GPIB::10", VI_FALSE, VI_FALSE, &vi);

viStatus = agpsa_SetAttributeViReal64(vi, "Trace1", AGPSA_ATTR_BANDWIDTH, 3E6);
viStatus = agpsa_SetAttributeViReal64(vi, "Trace1", AGPSA_ATTR_SPAN, 2E9);
```

Comparing IVI-COM and IVI-C

IVI-COM

- Collection interfaces indicate what functionality applies to repeated capabilities.

```
myNA.Window["a1"].Trace["S11"].Start=23;
```

IVI-C

- Need to know which attributes apply to a repeated capability and which apply to the driver as a whole.
- Nested repeated capabilities use an IVI-defined string-based syntax.

```
Acme12_WindowTraceStart(vi,"a1:S11",23)
```

Repeated Capability Access Pitfalls

```
// Wrong - must indicate which trigger repeated capability
ag34401_SetAttributeViReal64(session, VI_NULL,
        AG34401_ATTR_TRIGGER_LEVEL, 0.45);

// Wrong - Range applies to the whole driver, not to Channel1
ag34401_SetAttributeViReal64(session, "Channel1",
        AG34401_ATTR_RANGE, 100);

// Wrong - Enabled is a property of output repeated capability,
// not the trigger repeated capability
ag34401_SetAttributeViBoolean(session, "Out1:Trig1",
        AG34401_ATTR_OUTPUT_ENABLED, VI_TRUE);
```

Strings are not checked until runtime

Selecting Multiple Capabilities At Once

- Parameter used to specify repeated capability instances is known as a *repeated capability selector*
  - Same rules apply for all 3 repeated capability techniques
- *Simple repeated capability selector*
  - Single, non-nested repcap instance
  - May be a physical or virtual identifier
  - Example: “chan1”
- *Repeated capability ranges*
  - Lower bound to upper bound
  - Example: “1-3”, “8-10”
- *Repeated capability lists*
  - Simple comma-separated list
  - Example: “1, 4, 7, 9”
  - Combined Example: “1-3, 6, 8, 10-12”

- This demonstration shows
  - Collection style repeated capability in IVI-COM

Power supply with multiple outputs
Repeated capability allows controlling each output

- This demonstration shows
  - Selector style repeated capability in IVI-C
- Repeated Capability CVI Demo

Power supply with multiple outputs
Repeated capability allows controlling each output

End of Repeated Capabilities

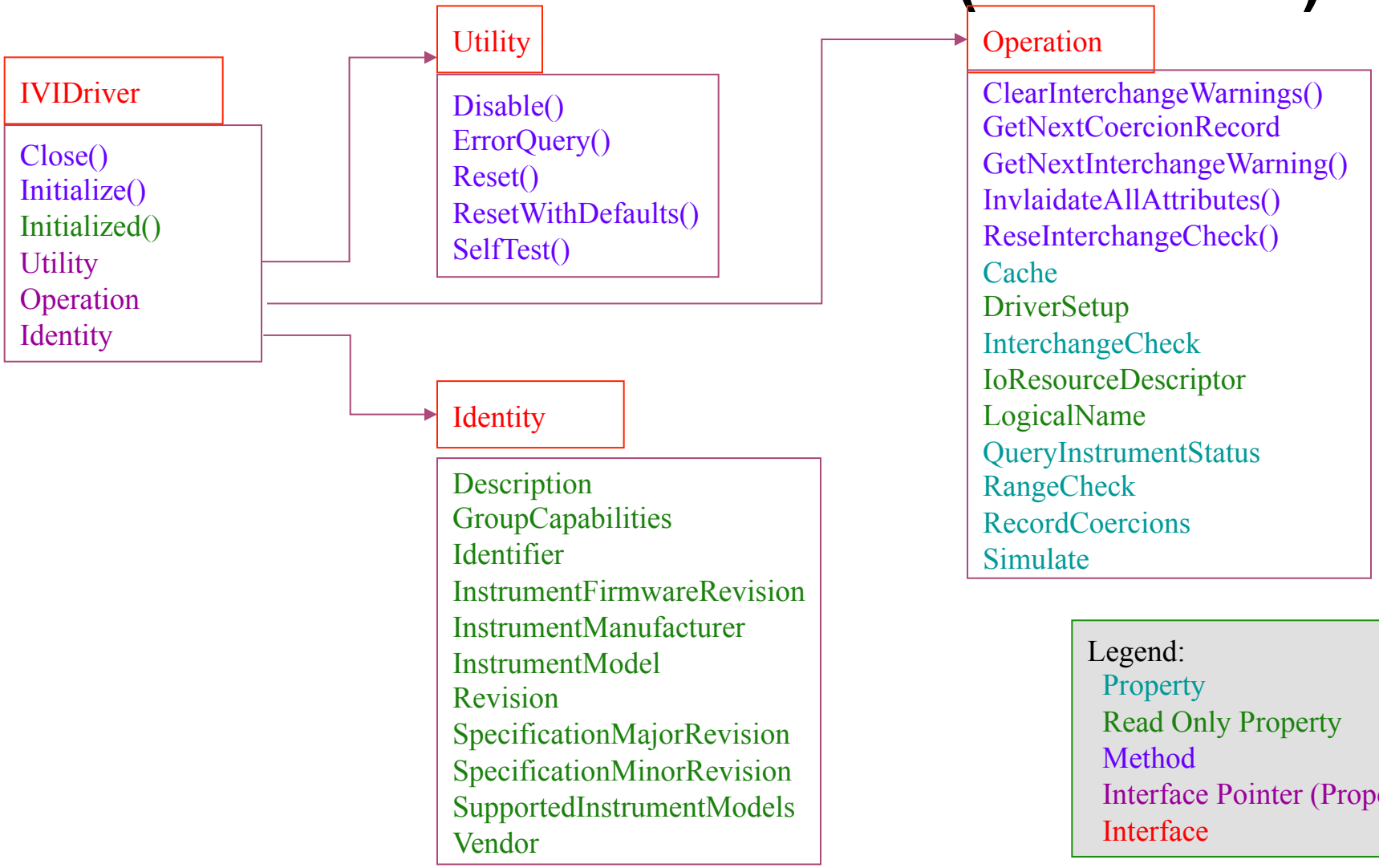
IVI Coding Patterns

Purpose: Show common features and how to use them
Help build a ‘mental model’ around IVI APIs

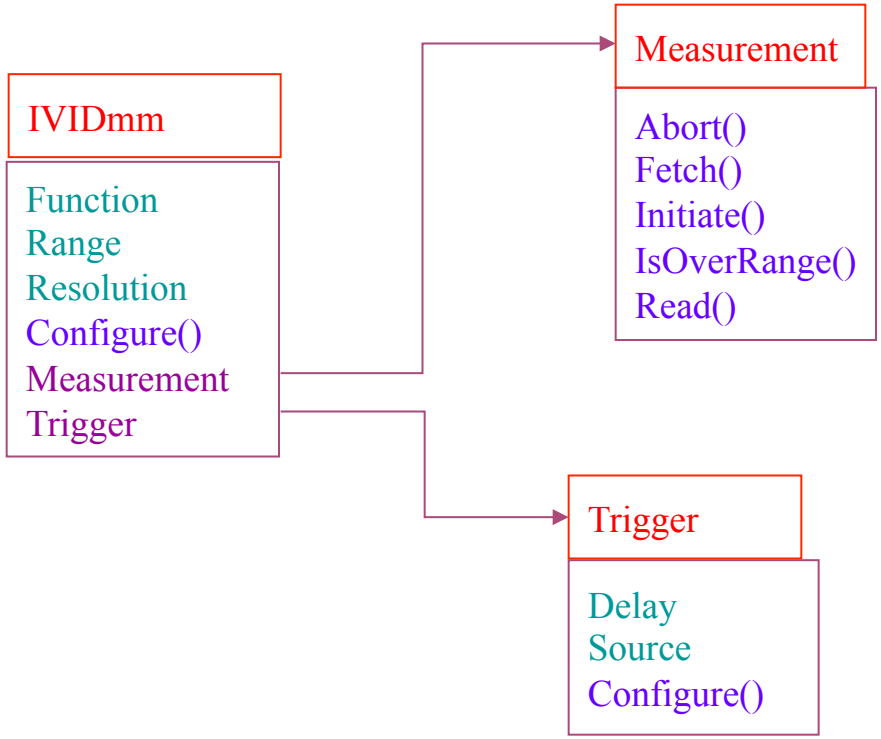
Topics:
- Using IVI Driver Features

Time: 00:30

Inherent IVI-COM/.NET API (all drivers)

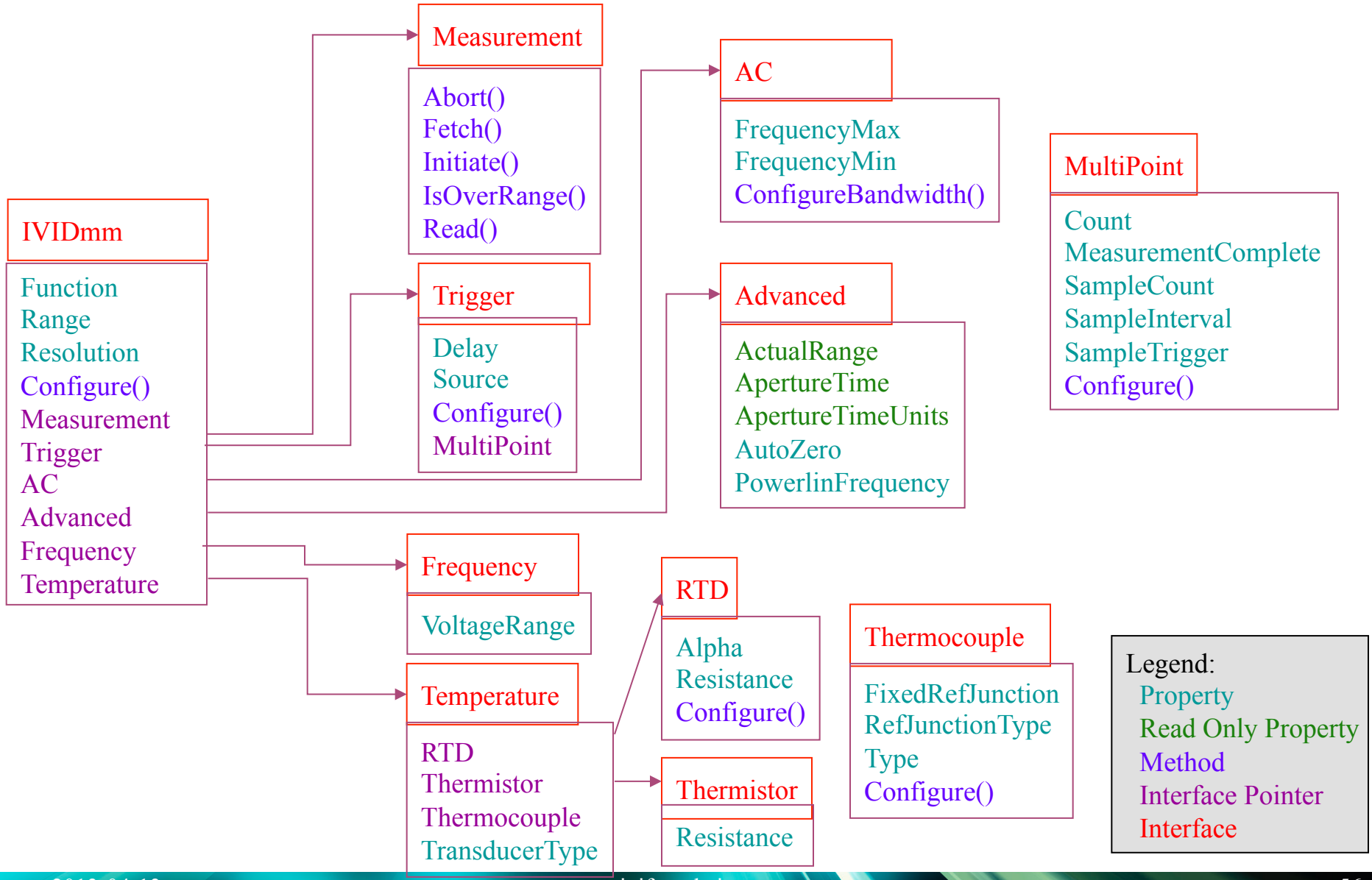


Base IVI-COM/.NET API for DMM



Legend:
- Property
- Read Only Property
- Method
- Interface Pointer (Property)
- Interface

Full IVI-COM/.NET API for DMM



Using the Hierarchy: Attributes

- Attributes provide:
  - Fine grain control
  - An easy way to read-back settings
- Important to:
  - Adjust a single setting independent of others
  - Faster if only a single parameter needs to be changed
  - May be necessary because not all attributes are in high-level config functions
- Examples:

```
C/COM: Dmm.Resolution = 10e-6;
C: IviDmm_SetAttributeReal64(Vi,
       IVIDMM_ATTR_RESOLUTION_ABSOLUTE,10e-6)
```

- Many important attributes have additional C functions

Using the Hierarchy: Configure

- Sets up a clump of the hierarchy in a single step

```
COM/NET:  dmm.Trigger.Configure(<trigger source>, <delay>)
C:        IviDmm_ConfigureTrigger
                          (Vi,TriggerSource,TriggerDelay)
```

- Set attributes in same location of hierarchy or lower
- Attributes not addressed can usually be left alone
- Programmer does not have to deal with couplings

```
COM/NET: dmm.Configure(<function>, <range>, <resolution>)
C:       IviDmm_ConfigureMeasurement
                        (Vi,Function,Range,Resolution);
```

Resolving Couplings with Configure

Following 2 blue boxes are nominally the same:

```
Dmm.Configure(AcmeFunctionEnum.AcmeFunctionDCVolts,23,10e-3)
// … make some measurements)
Dmm.Configure(AcmeFunctionEnum.AcmeFunctionOHMS, 1e6, 10);
// … make some measurements
```

```
Dmm.Range = 23;
Dmm.Resolution = 10e-3;
Dmm.Function = AcmeFunctionEnum.AcmeFunctionDCVolts;
// make some measurements
Dmm.Range = 1e6;
Dmm.Resolution = 10;
Dmm.Function = AcmeFunctionEnum.AcmeFunctionOhms;
// make some measurements
```

Resolving Couplings with Configure

```
Dmm.Configure(AcmeFunctionEnum.AcmeFunctionDCVolts,23,10e-3)
// … make some measurements)
Dmm.Configure(AcmeFunctionEnum.AcmeFunctionOHMS, 1e6, 10);
// … make some measurements
```

```
Dmm.Range = 23;
Dmm.Resolution = 10e-3;
Dmm.Function = AcmeFunctionEnum.AcmeFunctionDCVolts;
// make some measurements
Dmm.Range = 1e6;
Dmm.Resolution = 10;
Dmm.Function = AcmeFunctionEnum.AcmeFunctionOhms;
// make some measurements
```

When using properties, we attempt to set the voltage range to 1MVolt and get an error

General IVI Naming Conventions

- All instrument class names start with “Ivi”
  - Example: **IviScope**, **IviDmm**
- Function names
  - One or more words using Pascal casing
  - First word should be a verb

IVI-C Naming Conventions

- Function names
  - Class-compliant: <ClassName>_<FunctionName>
    - Example: **IviScope_ConfigureEdgeTriggerSource**
  - Instrument-specific: <DriverName>_<FunctionName>
    - Example: **ag54600_ConfigureEdgeTriggerSource**
- Attributes
  - All capitals
  - <CLASS_NAME>_ATTR_<ATTRIBUTE>
    - Example: **IVISCOPE_ATTR_TV_TRIGGER_SIGNAL_FORMAT**
  - Attribute values: <CLASS_NAME>_VAL_<VALUE>
    - Example: **IVISCOPE_VAL_EDGE_TRIGGER**

IVI-COM Naming Conventions

- Interface naming
  - Class compliant: Starts with “IIvi”
  - I<ClassName>
    - Example: **IIviScope**
  - Sub-interfaces add words to the base name that match the C hierarchy as close as possible
    - Examples: **IIviFgenArbitrary**, **IIviFgenArbitraryWaveform**
- Defined values
  - Enumerations and enum values are used to represent discrete values in IVI-COM
  - <ClassName><descriptive words>Enum
    - Example: **IviScopeTriggerCouplingEnum**
  - Enum values don’t end in “Enum” but use the last word to differentiate
    - Examples: **IviScopeTriggerCouplingAC** and **IviScopeTriggerCouplingDC**

Special Features of IVI Drivers

- Simulation
  - Driver provides basic operation when no instrument is present
- Range Checking
  - Drivers validate input values against instrument limits for an attribute
- State Caching
  - Driver caches values of attributes sent to/retrieved from instrument to improve performance by reducing I/O traffic
- Instrument Status Checking
  - Driver checks the status of the instrument at the end of each user operation
- Coercion and Coercion Recording
  - Driver coerces user-specified values to values accepted by the instrument
- Interchangeability Checking
  - Driver reports conditions it detects that may result in non-interchangeable behavior

Simulation (required feature)

```
dmm.Initialize("GPIB::10", False, False, "Simulate=True" )
‘ ... OR ...
dmm.DriverOperation.Simulate = True
‘default value is False
```

- Driver operates without creating an I/O session
- Allows user to begin working with test code before an instrument has been obtained
- IVI provides little guidance on how simulated data should be generated
  - Difficult to implement meaningful measurement behaviors
  - Ideal measurement simulation requires knowledge of the DUT response
- IVI Simulation extremely useful when developing system
  - Also provides a starting point for DUT simulation

Range Checking (required feature)

```
dmm.Initialize  “GPIB::10”, False, False, ”RangeCheck=True”
‘ ... OR ...
dmm.DriverOperation.RangeCheck = False
‘default value is True
```

- Driver validates parameters against allowed limits
  - IVI defines specific error codes to be returned for out of range values
- Valid ranges vary based on specific instrument model
- Difficult to implement in many instruments
  - Allowed ranges may be complex combination of configuration and measurement settings
- Many IVI drivers let the instrument perform the range checking
  - Attempt to set the value, then check if instrument succeeded

State Caching (optional feature)

```
dmm.Initialize “GPIB::10”, False, False, ”Cache=True”
‘ ... OR ...
dmm.DriverOperation.Cache = False
‘default value is True
```

- Goal is performance boost
  - Driver caches values sent to and retrieved from instrument in memory
  - Eliminates redundant I/O trips to instrument when settings are read/written more than once
- Very, very difficult to implement completely in complex instruments
  - Cache coherency problems occur easily
  - Instrument couplings abound
  - Coerced values complicate cache management
  - Driver may need to replicate significant portion of instrument algorithms
- Partial state caching frequently beneficial
  - Certain performance critical parameters may benefit
  - Must beware of coupled parameters
  - Most instruments also implement a cache internally, so be sure of the benefit

Instrument Status Checking (required feature)

```
dmm.Initialize “GPIB::10”, False, False, ”QueryInstrStatus=True”
‘ ... OR ...
dmm.DriverOperation.QueryInstrStatus = True
‘default value is False
```

- Required where possible (some instruments do not provide status check)
- Driver queries the instrument status at the end of each user operation that communicates with instrument
- Ensures instrument is operating as the driver expects
- Useful during implementation and debugging of application code
  - After validating the program, disable status checking to maximize performance

Coercion

- A continuous range of inputs, may need to be coerced into a limited set of legal values for the instrument
  - Example: User specifies a 25 Volt range for a DMM that has only 3, 30, 300, 3000 Volt ranges
- IVI class specifications define a coercion direction in which an IVI specific driver coerces a user-specified value
  - Up: Coerce to next value greater than user-specified value
  - Down: Coerce to next value less than user-specified value
  - None: Don't coerce => return an error if instrument can't be set to value
  - Driver allowed to coerce in different direction than spec suggests if the user's request can be satisfied

Coercion Recording (optional feature)

```
dmm.Initialize “GPIB::10”, False, False, ”RecordCoercions=True”
‘ ... OR ...
dmm.DriverOperation.RecordCoercions = True
‘default value is False
```

- Driver maintains in-memory log of coercions during a session
  - Internally a circular buffer of fixed size
  - Oldest value discarded when max size reached
  - Driver exports function for accessing the log

```
‘Returns an error if driver doesn’t implement coercion recording
HRESULT DriverOperation.GetNextCoercionRecord(
                                    [out, retval] BSTR* CoercionRecord);
```

I/O for IVI: GPIB and VXI Bus

- IVI drivers are required to use VISA for GPIB and VXI Bus
- Since IVI drivers use VISA, any driver works with any vendors GPIB or VXIBus hardware
  - IO Hardware vendor provides the VISA library
- This ensures the driver works with whatever GPIB or VXI solution the application uses
- Driver providers do not have to provide VISA library

I/O for IVI: Industry Standard IO

- IVI does not specify IO solution for other interfaces
  - Network (LAN/WAN)
  - IEEE 1394
  - USB
  - PCI/PCI Express
- These industry standard interfaces frequently do not require any special IO library, so IVI permits driver to use any solution
- These drivers usually include IO with the driver
  - VISA is still a common choice for T&M protocols
  - NOT an IVI Requirement

VISA APIs

- Both National Instruments and Agilent ship two VISA APIs as part of VISA
  - VISA-C (aka “classic VISA”, visa32.dll)
    - VPP-4.3.2: VISA Implementation Specification for Textual Languages
  - VISA-COM
    - VPP-4.3.4: VISA Implementation Specification for COM
- VISA.NET standards effort expected to complete 2014

Direct I/O Access with VISA

- VISA-COM

```
'This code uses the IFormattedIO488 interface returned
'from one of the driver properties
Dim dmm as New Agilent34401A
Dim io as IFormattedIO
dmm.Initialize "GPIB::10", False, False, ""

Set io = dmm.System.IO              'retrieve IFormattedIO488 interface
io.WriteString("SYST:CAL")
io.WriteNumber(13439)
'Other functions exist for reading/writing lists, IEEE binary blocks, etc.
```

- VISA-C

```
ViSession vi;
ViStatus viStatus = ag34401a_init("GPIB::10", VI_FALSE, VI_FALSE, &vi);
ViSession io = Ivi_IOSession(vi);
viPrintf (io, "SYST:CAL")

'Other functions exist for reading/writing lists, IEEE binary blocks, etc.
```

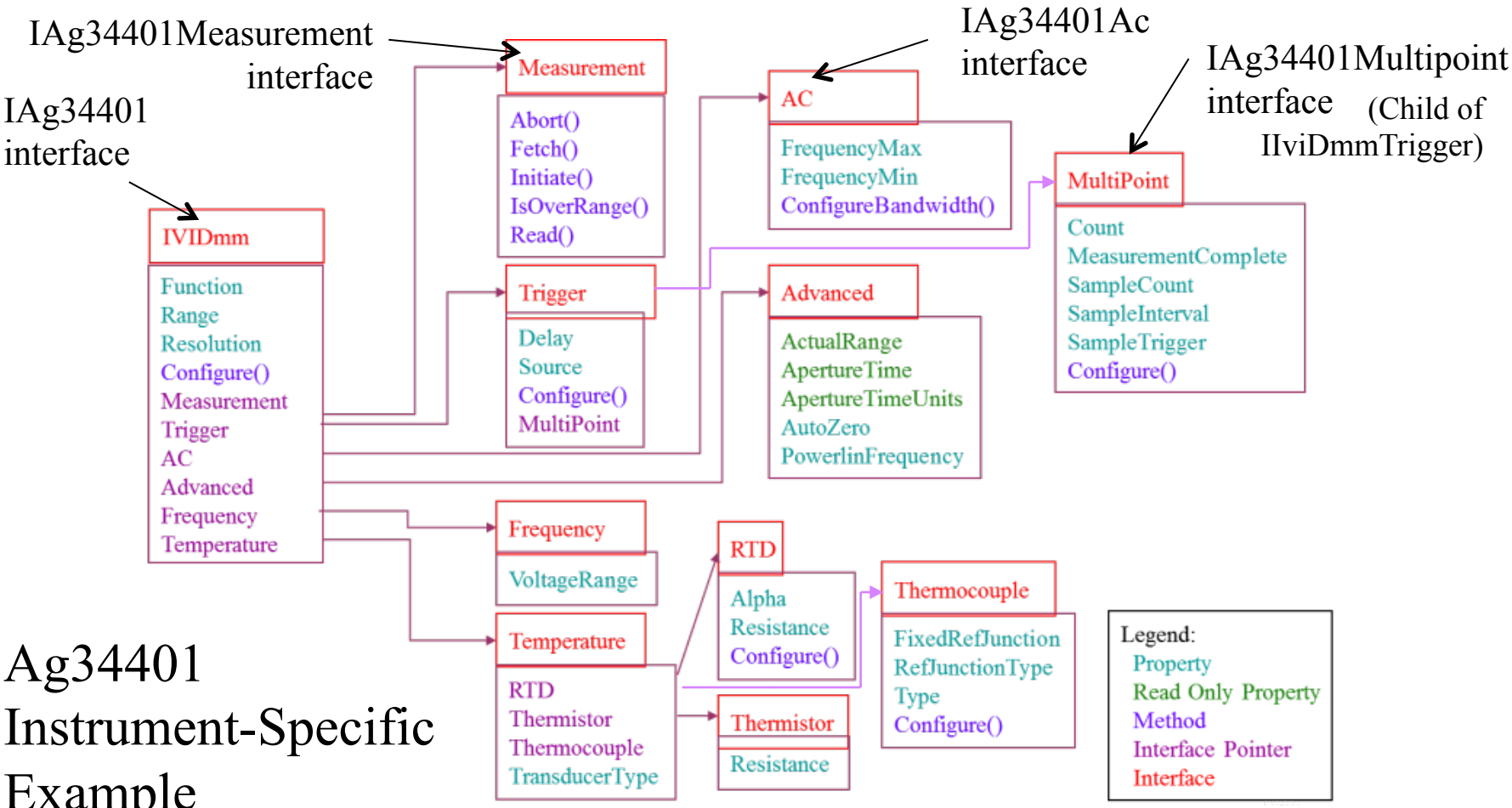
Direct I/O

- Useful for accessing functionality not supported by the driver
  - IVI drivers encouraged to expose the complete instrument capability
- Warning: Direct I/O access bypasses the driver completely
  - State-cached data may be invalidated
  - Other driver state may become invalid
- IVI-2014
  - IVI drivers for message-based instruments are required to provide a standardized direct I/O mechanism
  - API is similar to VISA Read and Write

Instantiating IVI-COM from .NET

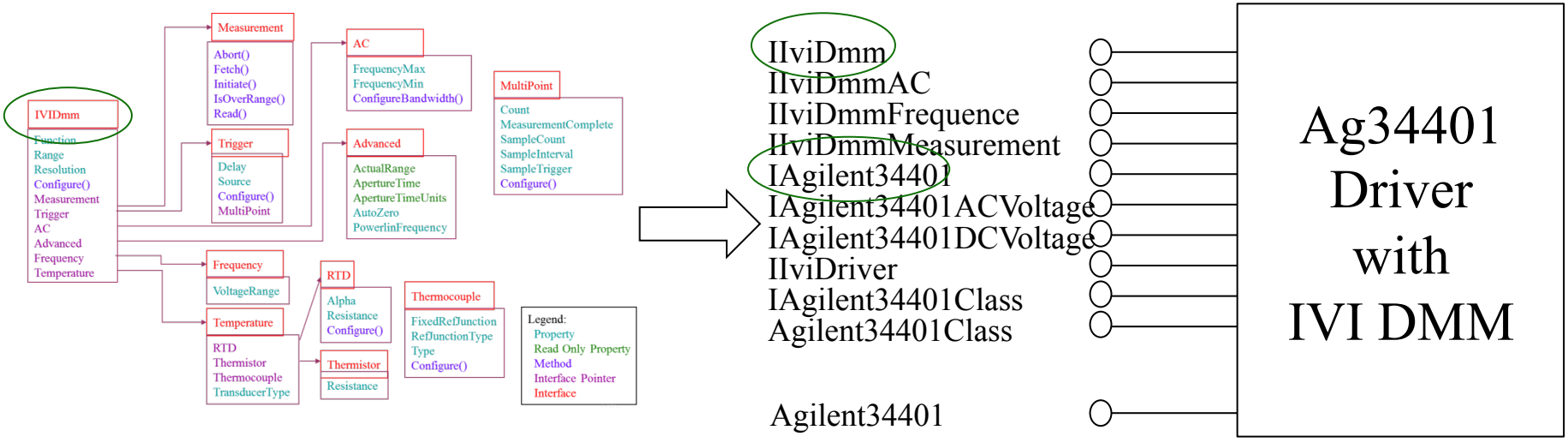
- .NET programs use a COM Interop (interoperability) assembly to call when calling IVI-COM from .NET.
  - This assembly is defined by IVI.
  - The Interop assembly code is generated with standard Microsoft tools which inspect the IVI-COM driver API to create the .NET wrapper.
- To understand what happens, we need to start by looking at the underlying COM object
- The core of every IVI-COM driver is a single object with many interfaces.
  - The interfaces are organized into two hierarchies.

The COM/.NET Hierarchy Implementation



Ag34401 Instrument-Specific Example

The Hierarchy Implementation



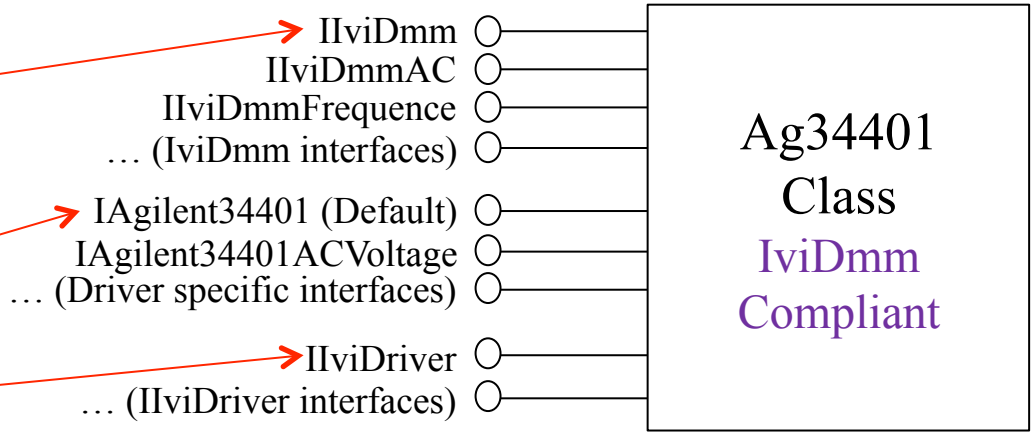
- The single driver object has many interfaces
- Interface reference pointers on each interface, implement the hierarchy:

```
myDmm.Temperature.RTD.Configure()
```

The Hierarchy Implementation

- The class-compliant hierarchy has IIviDmm at the root.
- The instrument-specific hierarchy has IAgilent34401 at the root (where Agilent34401 is the driver name).

(The IIviDriver interfaces are incorporated into both hierarchies.)



- IAgilent34401 contains references to child interfaces, which may in turn contain references to other child interfaces.
  - Collectively, these interfaces define the instrument-specific hierarchy.
  - IAgilent34401, the default, is what you get when you directly instantiate the driver.
- IIviDmm contains references to child interfaces, which may in turn contain references to other child interfaces.
  - Collectively, these interfaces define the class compliant hierarchy.

.NET Interop Types to Choose From

Type of the driver pointer

IIviDmm	Root of class compliant interface
IAgilent34401	Root of instrument specific interface
Agilent34401	Default interface of Agilent34401Class » Can be used to instantiate the driver » Corresponds to Iagilent34401
Agilent34401Class	Class with ALL interfaces

Class to Instantiate

Agilent34401 is the only type instantiable across all versions of the .NET Framework

IVI-COM Interop Type Demonstration

- This demonstration shows
  - How different data types work with the driver

1. Why the intuitive syntax:

```
var Driver = new Agilent34410Class();
```

   Can be troublesome

2. Getting the IviDmm hierarchy from device specific API

```
static void Main(string[] args)
{
   Agilent34410       myDmm1 = new Agilent34410();        // PREFER:Like IVI.NET
   IAgilent34410      myDmm2 = new Agilent34410();        // PREFER:Explicit
   Agilent34410Class  MyDmm3 = new Agilent34410Class();   // CAUTION!
   Agilent34410       myDmm4 = new Agilent34410Class();   // CAUTION! FW Version

   IIviDriver myDriver = myIviDmm as IIviDriver;    //Gets IVI Driver intfc
   IIviDmm    MyDmm5   = (IIviDmm)myDmm1;           //Also gets IVI DMM root

   myDmm.Diode.Configure();
   MyDmm3.RTD.Resistance = 42;              // Doesn't reflect hierarchy
   MyDmm5.Configure(IviDmmFunctionEnum.IviDmmFunction2WireRes, ....);
}
```

IVI-C and IVI .NET

- IVI-C DLL does not have a hierarchy
  - Each function has a unique name in a flat name space.
  - A separate file provides a hierarchy for the tools that can read it (NI CVI)

- IVI.NET has similar structure to IVI-COM
  - The instrument specific classes are: <Product> (e.g., Agilent34401)
  - The IVI defined classes are: <IviType> (e.g., IviDmm)
  - The root interface for the hierarchy: I<Product> (e.g., IAgilent34401)
- The preferred method for instantiating a .NET driver is:

```
Agilent34410 myDmm = new Agilent34410();
```

(when interchangeability is not a goal)

End of IVI Coding Patterns

IVI Advanced Topics

Purpose: Overview of other IVI Driver topics

Topics:

- Shared Components
- Config Store, Config Server
- Session Factory
- C Shared Components
- Interchangeability with Config Store
- Locking

Time: 00:45

IVI Shared Components

- IVI Foundation develops and maintains several software components for performing common operations in IVI-based test systems
- Different member companies provide the development horsepower
- Shared Component Installer also provided by Foundation
  - Must be used by driver vendors to install Shared Components
  - Illegal for driver suppliers to install Shared Components without it

IVI Shared Components (cont)

- IVI Configuration Store
  - Central repository for driver configuration and initialization data
- IVI Configuration Server
  - Software component(s) for accessing IVI Configuration Store
- COM Session Factory
  - COM component for instantiating IVI-COM drivers in an interchangeable fashion
- C Shared Components
  - Variety of service components for building IVI-C drivers

IVI Configuration Store

- XML file installed with the IVI Shared Components
  - <IviInstallDir>\Data\IviConfigurationStore.xml
- Fairly complex schema
  - IviConfigurationStore.xsd
- Highly self-referential
  - *Manual editing officially allowed but discouraged*
- Read/written by drivers and end-users
- Extensible
  - Supports custom driver, application or end-user data

IVI Configuration Server

- Software component(s) developed and maintained by the IVI Foundation
- COM component
  - <IviInstallDir>\Bin\IviConfigServer.dll
- C API
  - <IviInstallDir>\Bin\IviConfigServerCAPI.dll
- Recommended way of accessing IVI Configuration Store

Using the IVI Configuration Store

- Typical uses
  - Driver initialization data, such as the I/O address
  - Virtual-to-physical repcap name mapping => needed for interchangeability
  - Logical name mappings => needed for interchangeability
  - Custom driver features
- Application related information
  - Associated modular instruments into a meta-instrument

Primary Elements in the Config Store

- Software Module
  - Describes a software component (.dll) installed on the system
  - This represents “the driver”
  - Created by driver installers
- Hardware Asset
  - Describes a physical instrument
  - Not typically created by driver installers
  - Main purpose is to abstract away the I/O address
- Driver Session
  - Represents an association of a Software Module with a Hardware Asset
  - Houses property settings that control driver behavior (state-caching, range-checking, coercion, error checking, etc.)
  - May or may not be created by a driver installer (NI does, Agilent doesn’t)
- Logical Name
  - User-specified name
  - Never created by driver installers

COM Session Factory

- Very simple COM component
- Required for full interchangeability
  - Used in client code, not driver code
  - Gets reference to a specific model of instrument out of initialization code
  - No changes to client code required when changing IVI-COM drivers and Session Factory is in use
- "Create by Name"
  - User-defined LogicalName mapped to specific driver in Configuration Store
  - Session Factory calls CoCreateInstance on the correct specific driver
  - No direct references to a specific driver anywhere in client code
- Described in detail in *IVI-3.6: COM Session Factory Specification*

```
// Defined on IIviSessionFactory interface
HRESULT CreateDriver([in] BSTR LogicalName, [out, retval] IUnknown** Driver);
```

Using the COM Session Factory

```
Dim factory as New IviSessionFactory
Dim scope as IIviScope
Set scope = factory.CreateDriver("MyScope")   'Factory looks up "MyScope" in
                                               'IVI Configuration Store and
                                               'creates specific driver to which
                                              'it is mapped
scope.Initialize "MyScope",False,False,""
scope.Acquisition.NumberOfPointsMin = 1000
```

IVI-C Shared Components

- Described in detail in *IVI-3.9: C Shared Components Specification*
- Dynamic Driver Loader
  - Used by class drivers to dynamically load class-compliant specific drivers
- Error Message Component
  - Helps driver developers create error messages
- Session Management Component
  - Creating and destroying IVI driver sessions
  - Managing session instance data and mulithreaded locks
- Session Error Component
  - Provides access to per-session errors
- Multithread Lock Component
  - Acquiring and releasing locks
- Thread-Local Storage Component
  - Provides access to thread-based error information

C Shared Components and Intended Users

Component	Intended Users
Dynamic Driver Loader	**Class Driver**
Error Message	**Specific Driver, Class Driver**
Session Management	**Specific Driver, Class Driver**
Session Error	**Session Management Component**
Multithread Lock	**Session Management Component**
Thread Local Error Storage	**Session Management Component**
Thread Local Storage	**Thread Local Error Storage Component**

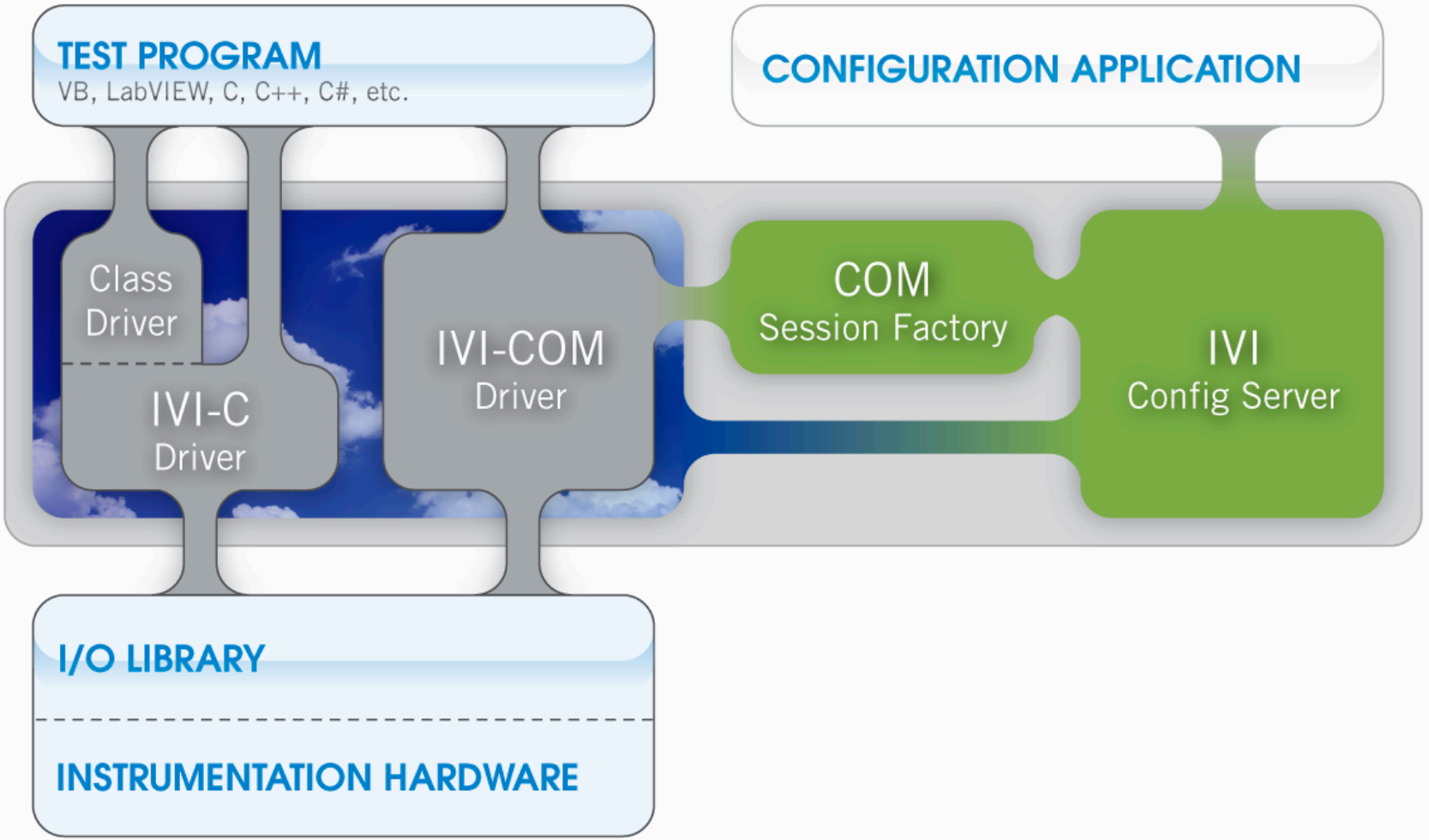
Session Management Component

Function	Purpose
IviSession_New	**Creates an IVI driver session.**
IviSession_SetDataPtr	**Associates a pointer to instance data with a session.**
IviSession_GetDataPtr	**Retrieves a pointer to instance data from a session.**
IviSession_Lock	**Obtains a multithreaded lock on a session.**
IviSession_Unlock	**Releases a multithreaded lock on a session.**
IviSession_SetError	**Sets the error information for an IVI session and for the current execution thread.**
IviSession_GetError	**Retrieves and clears the error information for the session or the current execution thread.**
IviSession_ClearError	**Clears the error information for the session or the current execution thread.**
IviSession_Dispose	**Closes an IVI driver session.**

Error Message Component

Function	Purpose
IviErrorMessage_Get	**Retrieves the static message associated with a specific error code.**
IviErrorMessage_FormatWithElaboration	**Formats an error description from two error messages and places it into an output buffer.**

IVI Interchangeability Architecture



Interchangeability with IVI-COM

- Requirements for the driver developer
  - Implement one or more class-compliant interfaces
- Requirements for the client programmer
  - Use the COM Session Factory to instantiate the driver
    - This involves setting up a logical name in the Configuration Store
  - Use only the class-compliant interfaces in the sections of code which are to be interchangeable
  - Use only virtual repeated capability identifiers
    - Involves setting up virtual-to-physical mappings in the Configuration Store
- Client code contains no references to any specific driver
- No proprietary components are required

Non-Interchangeable IVI-COM Code

```
'This code is not interchangeable for a number of reasons

Dim scope as New Agilent54600

scope.Initialize "GPIB::10", False, False, ""

scope.Channels("Chan1").Range = 10.0
scope.Channels("Chan1").Offset = 1.5

scope.Close
```

Non-Interchangeable IVI-COM Code

```
'This code is not interchangeable for a number of reasons

Dim scope as New Agilent54600              // refer to model

scope.Initialize "GPIB::10", False, False, ""  // location

scope.Channels("Chan1").Range = 10.0    // "Chan" name is
scope.Channels("Chan1").Offset = 1.5    //  not in IVI spec

scope.Close
```

Syntactically Interchangeable Code

```
'This code is interchangeable
'No references to a specific driver, device, or channel name
Dim factory as New IviSessionFactory
Dim scope as IIviScope
Set scope = factory.CreateDriver("MyScope")        'use Session Factory with
                                                   'logical name

scope.Initialize "MyScope", False, False, ""       'logical name instead of
                                                   'fixed GPIB address

scope.Channels("MyChan").Range = 10.0              'virtual channel name
scope.Channels("MyChan").Offset = 1.5              'virtual channel name

scope.Close
```

Interchangeability with IVI-C

- Requirements for the driver developer
  - Implement a class-compliant capability group (only 1 class allowed)
- Requirements for the client programmer
  - Download a class driver from National Instruments
  - Use class driver Initialize function with logical name to create a session
    - Involves setting up a logical name in the Configuration Store
  - Use only the class driver APIs in the client application
  - Use only IVI-defined attribute identifiers and attribute values
  - Use only virtual repeated capability identifiers
    - Involves setting up virtual-to-physical mappings in the Configuration Store
- Client code contains no references to any specific driver
- National Instruments class driver components required

Non-Interchangeable IVI-C Code

```
// This code is not interchangeable
#include “ag54600.h”                  // direct reference to driver
                                     // also links with “ag54600.lib”
int _tmain(int argc, _TCHAR* argv[])
{
   // Functions and attributes from specific driver are used
   ViSession vi;
   ViStatus viStatus = ag54600_init(“GPIB::10”, VI_FALSE, VI_FALSE, &vi);

   viStatus = ag54600_SetAttributeViReal64(vi, “Chan1” /*physical name*/,
     AG54600_ATTR_VERTICAL_RANGE /*specific attribute*/, 10.0);

   viStatus = ag54600a_SetAttributeViInt32(vi, “Chan1” /*physical name*/,
     AG54600_ATTR_TRIGGER_TYPE /*specific attribute*/,
     AG54600_VAL_EDGE_TRIGGER /*specific attribute value*/ );

   viStatus = ag54600_close();
}
```

Interchangeable IVI-C Code

```
// This code is syntactically interchangeable
#include “iviscope.h”            // class driver header file
                              // also link with iviscope.lib
int _tmain(int argc, _TCHAR* argv[])
{
   // Only functions and attributes from class  driver are used
   ViSession vi;
   ViStatus viStatus = IviScope_init(“MyScope”, VI_FALSE, VI_FALSE, &vi);

   viStatus = IviScope_SetAttributeViReal64(vi, “MyChan” /*virtual name*/,
      IVISCOPE_ATTR_VERTICAL_RANGE /*IVI attribute*/, 10.0);

   viStatus = IviScope_SetAttributeViInt32(vi, “MyChan” /*virtual name*/,
      IVISCOPE_ATTR_TRIGGER_TYPE /*IVI attribute*/,
      IVISCOPE_VAL_EDGE_TRIGGER /*IVI attribute value*/ );

   viStatus = IviScope_close();
}
```

When Interchangeability Isn't

- Syntactically interchangeable code is not guaranteed to produce the same answer
  - *syntactic interchangeability* versus *semantic interchangeability*
- Different answers from syntactically interchangeable code can come from a variety of sources
  - Different measurement techniques
  - Different default values for state variables
  - Different instrument specifications
  - Timing differences
- The more complex the instrument, the more difficult it can be to achieve semantic interchangeability

When Interchangeability Isn't

- Syntactically interchangeable code is not guaranteed to produce the same answer
  - *syntactic interchangeability* versus *semantic interchangeability*
- Different answers from syntactically interchangeable code can come from a variety of sources
  - Different measurement techniques
  - Different default values for state variables
  - Different instrument specifications
  - Timing differences
- The more complex the instrument, the more difficult it can be to achieve semantic interchangeability

Interchangeability Checking (optional feature)

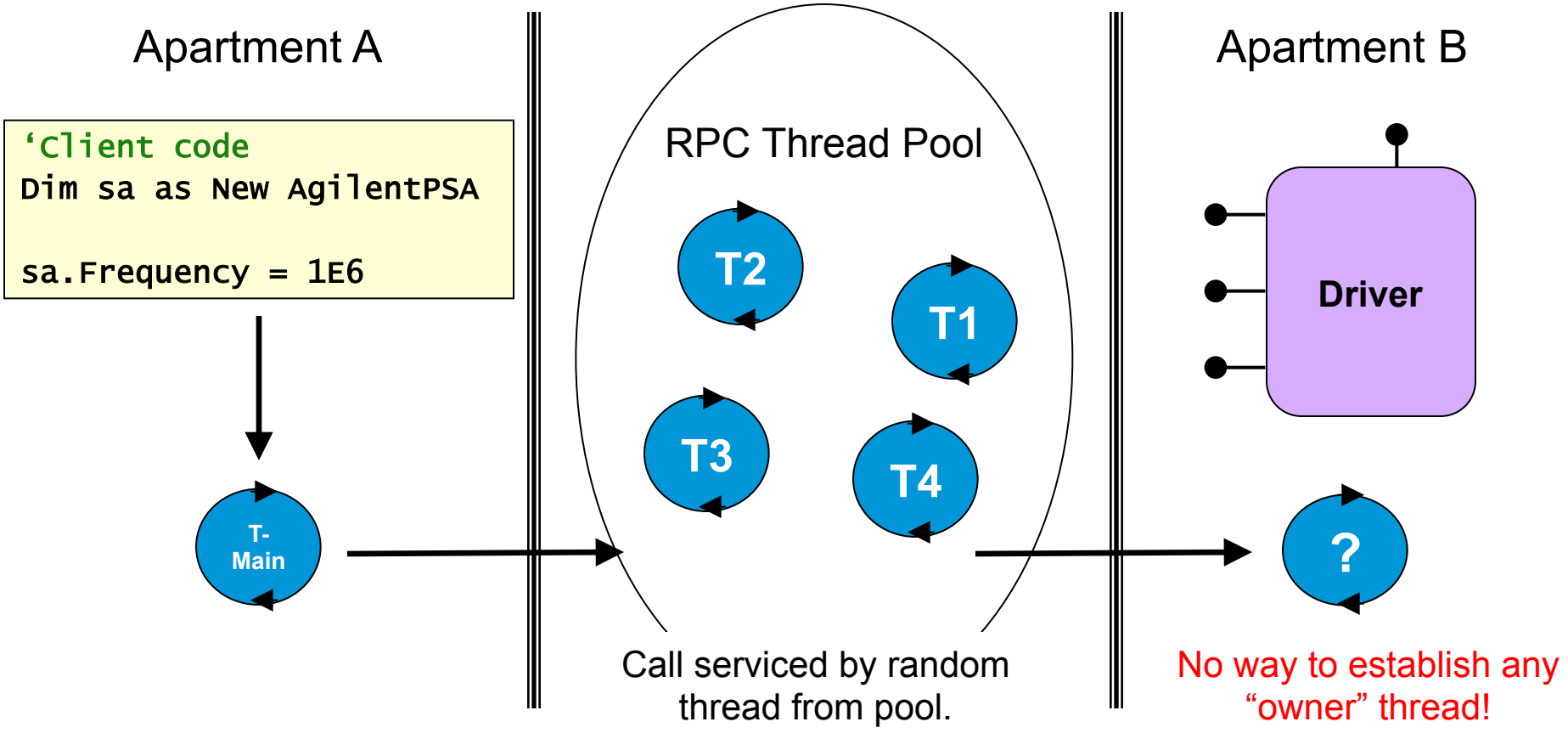
```
dmm.Initialize “GPIB::10”, False, False, ”InterchangeCheck=True”
‘ ... OR ...
dmm.DriverOperation.InterchangeCheck = True
‘default value is False
```

- Identifies where a client program is using default values
  - Warnings generated when program doesn’t fully specify the configuration
  - It is a poor practice to override all defaults in highly complex instruments
  - Properly chosen defaults may enhance interchangeability
- Potentially useful for simple instruments during debug
  - Frequently not implemented

Multithread Safety

- IVI drivers must be operate correctly in a multithreaded environment
  - IVI isn't specific about what level of locking
    - Object-level? Method-level? Interprocess locking?
- Thread safety and locking is often confused with resource locking
  - Resource locking protects the instrument from simultaneous access
  - IVI does not address resource locking
- IVI-COM drivers must use the "Both" apartment threading model
  - Allows driver to live in same apartment as client
  - Involves protecting all instance data with a synchronization lock
- Confusingly, LockObject and UnlockObject functions are required by the IVI specifications to return an error
  - Problem in spec discovered (by us) after approval
  - Not possible to implement locks between method calls in COM
- IVI-C drivers ensure correct operation of asynchronous calls to driver sessions by multiple threads in the same process

Locking Between Method Calls in COM



End of IVI Advanced Topics

Follow-up

- For more information
  - IVI Website: www.ivifoundation.org
  - IVI Getting Started guides: www.ivifoundation.org
  - IVI Specifications: www.ivifoundation.org
  - IVI Registration page: www.ivifoundation.org
- Most vendors have documentation and drivers on their website
- For questions on these slides, contact
  - Kirk Fertitta kirk@pacificmindworks.com
  - Joe Mueller joe_mueller@agilent.com

IVI Organization and Purpose

Purpose:

- Describe the IVI Organization
- Describe what an IVI driver is

Topics:

-

Time: 0:15

Goals of the IVI Foundation

Quality

- To improve driver quality
- To establish guidelines for driver testing and verification
- Establish a consistent architectural framework for multi-vendor systems

Hardware Interchangeability

- To simplify the task of replacing an instrument from a system with a similar one
- To preserve test software when instruments become obsolete
- To simplify test code reuse from design validation to production test

Software Interchangeability

- To provide an architectural framework that allows users to easily integrate software from multiple vendors
- To provide standard access to driver capabilities such as simulation, state caching, range checking and coercion
- To provide consistent instrument control in popular programming environments

IVI Driver Standards

- Architecture specs
- Requirement for all drivers
- Ensures all work together
- Important for any driver
- Common functionality
- Common components
- Common style
- Installation
- Driver types: C/COM/.NET

- Class specs
- Requirements for a type of instrument
- Provides syntactic interchangeability
- Establishes common paradigms for consistency
- Limited to common functionality

Driver Architecture Specifications

- IVI-3.1: Driver Architecture Specification
- IVI-3.2: Inherent Capabilities Specification
- IVI-3.3: Standard Cross-Class Capabilities Specification
- IVI-3.4: API Style Guide
- IVI-3.5: IVI Configuration Server Specification
- IVI-3.6: COM Session Factory Specification
- IVI-3.9: C Shared Components
- IVI-3.12: Floating Point Services Specification
- IVI-3.14: Primary Interop Assembly Specification
- IVI-3.15: IviLxiSync Specification
- IVI-3.17: Installer Requirements Specification
- IVI-3.18: IVI.NET Utility Classes and Interfaces Specification

IVI Instrument Classes

- DC power supply
- AC power supply
- DMM
- Function generator
- Oscilloscope
- Power meter
- RF signal generator
- Spectrum analyzer
- Switch
- Upconverter
- Downconverter
- Digitizer
- Counter/timer

IVI Driver Features

- Simulation
  - Required of all IVI drivers
  - Very useful for testing in new ADEs
  - Helpful when instruments are difficult to procure
- State caching
- Range checking
- Coercion
  - Coercion recording
- All extended features enabled and accessed in a standard fashion

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 Compliance

- IVI Compliant – Follows Architecture Specs
  - Installation (IVI-3.17)
  - Inherent Capabilities (IVI-3.2)
  - Cross Class Capabilities (IVI-3.3)
  - Style (IVI-3.4)
  - Custom instrument API – complies with IVI-3.4, 3.3, 3.1
- IVI Class Compliant – Implements Defined Class
  - Also IVI compliant
  - Provides class API in addition to Custom API
  - Custom API may be omitted (unusual)

Why IVI for Vendors? – Track Evolving Customer Needs

- Vendors relieved from onerous task of keeping pace with multiple moving targets
  - Windows OSs, Windows Help, Windows Installer, Windows API, security, .NET platform
  - Six versions of Visual Studio since IVI
  - Vista and UAC complex changes
  - Parallel support for 32-bit and 64-bit OS also complex
    - IVI provides uniform and complete solutions
- IVI member companies bring experts to meetings to ensure IVI solutions work with their hardware
  - Users of IVI directly leverage R&D efforts of NI, Agilent, R&S, etc.

Why IVI for Vendors

- Several IVI tools are available to facilitate the creation of IVI drivers
- IVI tools can help you with:
  - Creating driver shell that complies with IVI inherent capabilities and IVI class-defined capabilities
  - Creating and modification of attributes
  - Creating help files
  - Creating installers
    - IVI-3.17 spec dedicated to IVI installers and is *53 pages*
  - Creating unit test and regression tests
  - Creating special components for .NET, COM, or C
    - .NET: XML IntelliSense file, interop assembly, version policy files
    - C: function panels

IVI Membership Benefits

- Influence the development of standards
- Participate in and access future standards
- Share ideas with developers, users, system integrators and vendors
- Access source code for shared components
- Participate in interoperability sessions
- Network with test and measurement industry leaders

Architectural Approachs for Interchangeability

Purpose:

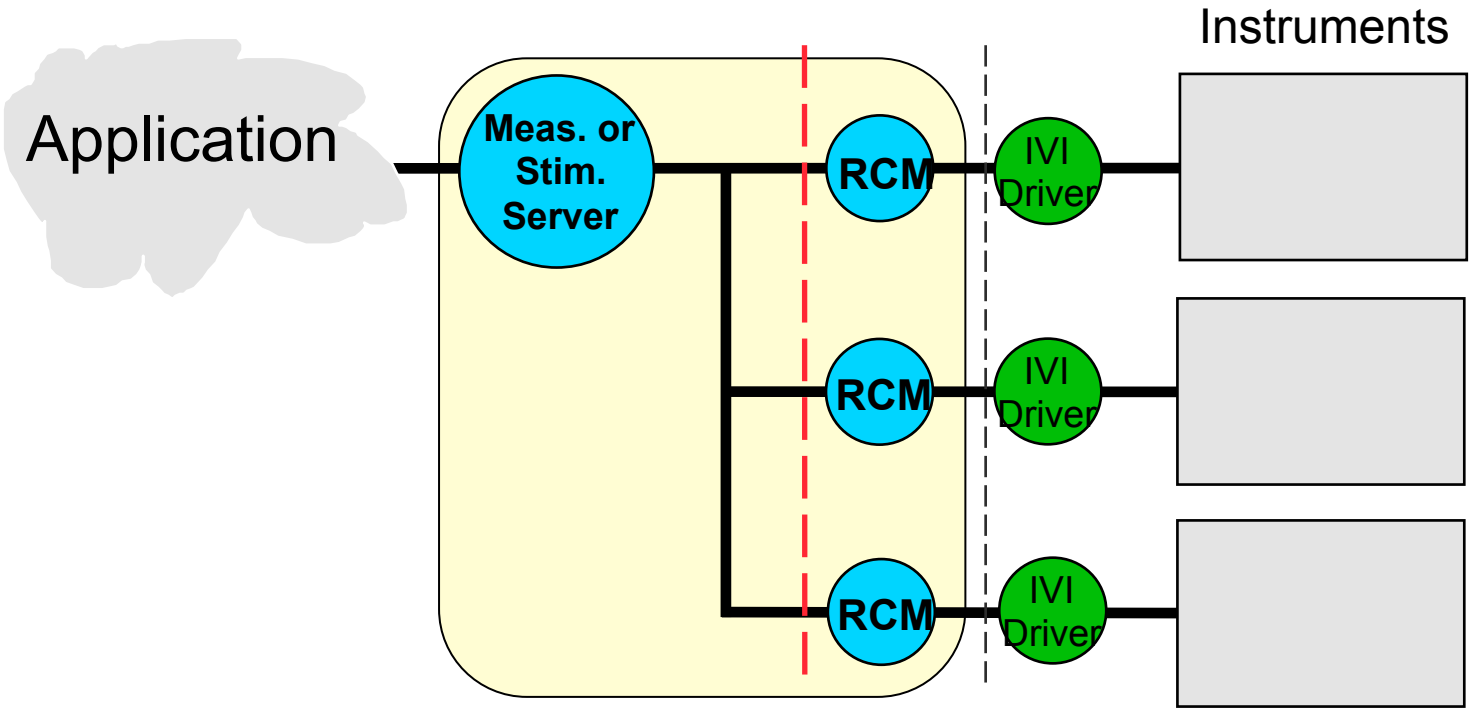
- Review architectural steps for greater interchangeability (IVI-MSS)

Topics:

- IVI-MSS

Time: 00:15

IVI-MSS: Protect the Measurement from Instrument Variations



RCM (Role Control Module)
- API defined by *Measurement*
- Hides most of instrument from measurement
- Enables porting measurement to any instrument, only requiring limited functionality