

Keysight M9241/42/43A PXIe Oscilloscopes

Notices

© Keysight Technologies, Inc. 2017

No part of this manual may be reproduced in any form or by any means (including electronic storage and retrieval or translation into a foreign language) without prior agreement and written consent from Keysight Technologies, Inc. as governed by United States and international copyright laws.

Manual Part Number

M9240-97003

Edition

First edition, January 2017

Printed in Malaysia

Published by:
Keysight Technologies, Inc.
1900 Garden of the Gods Road
Colorado Springs, CO 80907 USA

Print History

M9240-97003, January 2017

Warranty

The material contained in this document is provided "as is," and is subject to being changed, without notice, in future editions. Further, to the maximum extent permitted by applicable law, Keysight disclaims all warranties, either express or implied, with regard to this manual and any information contained herein, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. Keysight shall not be liable for errors or for incidental or consequential damages in connection with the furnishing, use, or performance of this document or of any information contained herein. Should Keysight and the user have a separate written agreement with warranty terms covering the material in this document that conflict with these terms, the warranty terms in the separate agreement shall control.

Technology License

The hardware and/or software described in this document are furnished under a license and may be used or copied only in accordance with the terms of such license.

U.S. Government Rights

The Software is "commercial computer software," as defined by Federal Acquisition Regulation ("FAR") 2.101. Pursuant to FAR 12.212 and 27.405-3 and Department of Defense FAR Supplement ("DFARS") 227.7202, the U.S. government acquires commercial computer software under the same terms by which the software is customarily provided to the public. Accordingly, Keysight provides the Software to U.S. government customers under its standard commercial license, which is embodied in its End User License Agreement (EULA), a copy of which can be found at www.keysight.com/find/sweula. The license set forth in the EULA represents the exclusive authority by which the U.S. government may use, modify, distribute, or disclose the Software. The EULA and the license set forth therein, does not require or permit, among other things, that Keysight: (1) Furnish technical information related to commercial computer software or commercial computer software documentation that is not customarily provided to the public; or (2) Relinquish to, or otherwise provide, the government rights in excess of these rights customarily provided to the public to use, modify, reproduce, release, perform, display, or disclose commercial computer software or commercial computer software documentation. No additional government requirements beyond those set forth in the EULA shall apply, except to the extent that those terms, rights, or licenses are explicitly required from all providers of commercial computer software pursuant to the FAR and the DFARS and are set forth specifically in writing elsewhere in the EULA. Keysight shall be under no obligation to update, revise or otherwise modify the Software. With respect to any technical data as defined by FAR 2.101, pursuant to FAR 12.211 and 27.404.2 and DFARS 227.7102, the U.S. government acquires no greater

than Limited Rights as defined in FAR 27.401 or DFAR 227.7103-5 (c), as applicable in any technical data.

Safety Notices

CAUTION

A **CAUTION** notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in damage to the product or loss of important data. Do not proceed beyond a **CAUTION** notice until the indicated conditions are fully understood and met.

WARNING

A **WARNING** notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in personal injury or death. Do not proceed beyond a **WARNING** notice until the indicated conditions are fully understood and met.

Contents

1 What You Will Learn in This Programming Guide

Related Websites / 6

Related Documentation / 7

Overall Process Flow / 8

2 Installing Hardware, Software, and Licenses

3 APIs for the M9241/42/43A PXIe Oscilloscopes

IVI Compliant or IVI Class Compliant / 13

IVI Driver Types / 14

IVI Driver Hierarchy / 16

Class-Complaint and Instrument-Specific Hierarchies for the M924xA / 17

Naming Conventions Used to Program IVI Drivers / 19

General IVI Naming Conventions / 19

IVI-COM Naming Conventions / 19

4 Creating a Project with IVI-COM Using C-Sharp

Step 1 - Create a Console Application / 22

Step 2 - Add References / 23

Step 3 - Add "using" Statements / 25

Step 4 - Create Instance of the IVI-COM Driver / 26

Step 5 - Initialize the Driver Instance / 27

Resource Names / 27

Initialize() Parameters / 29

Initialize() Options / 30

Step 6 - Write the Program Steps / 33

Step 7 - Close the Driver / 34

Step 8 - Building and Running a Complete Program Using Visual C-Sharp / 35

Example Program - Code Structure / 35

Example Program - Full Code Listing / 36

5 Creating a Project with IVI-COM Using Python

6 References

7 Glossary

Index

1 What You Will Learn in This Programming Guide

Related Websites / 6

Related Documentation / 7

Overall Process Flow / 8

This programming guide is intended for individuals who write and run programs to control test-and-measurement instruments. Specifically, in this programming guide, you will learn how to use Visual Studio 2010 with the .NET Framework to write IVI-COM Console Applications in Visual C#. Knowledge of Visual Studio 2010 with the .NET Framework and knowledge of the programming syntax for Visual C# is required.

Our basic user programming model uses the IVI-COM driver directly and allows customer code to:

- Access the IVI-COM driver at the lowest level
- Control the Keysight M9241/42/43A PXIe oscilloscopes

This guide describes:

- Example Program: How to Print Driver Properties, Check for Errors, and Close Driver Sessions

Additional example programs show how to perform waveform acquisitions.

Related Websites

- [Keysight Technologies PXI and AXIe Modular Products](#)
 - [Keysight M9241/42/43A PXIe Oscilloscopes](#)
- [Keysight Technologies](#)
 - [IVI Drivers & Components Downloads](#)
 - [Keysight I/O Libraries Suite](#)
 - [GPIB, USB, & Instrument Control Products](#)
 - [Keysight VEE Pro](#)
 - [Technical Support, Manuals, & Downloads](#)
 - [Contact Keysight Test & Measurement](#)
- [IVI Foundation](#) - Usage Guides, Specifications, Shared Components Downloads
- [MSDN Online](#)

Related Documentation

To access documentation related to the Keysight M9241/42/43A PXIe oscilloscope modules and the Keysight M9240A AutoProbe power module, use one of the following methods:

- The related documents are available on the product CD:

Document	Description	File name	Format
Startup Guide	Includes procedures to help you to unpack, inspect, install (hardware and software), verify operation, and make a basic measurement.	M924x_StartupGuide.pdf	PDF
Soft Front Panel (SFP) User's Guide	Shows how to use the InfiniiVision M9241/42/43A PXIe oscilloscope's Soft Front Panel (SFP) user interface.	M924x_SFP_Users_Guide.pdf	PDF
		M924x_SFP_Users_Guide.chm	CHM (Microsoft Help Format)
SCPI Programmer's Guide	Shows how to program the M9241/42/43A PXIe oscilloscopes using SCPI commands.	M924x_SCPI_Programmers_Guide.chm	CHM (Microsoft Help Format)
		M924x_SCPI_Programmers_Guide.pdf	PDF
IVI Programming Guide (this manual)	Shows you how to use Visual Studio 2010 with the .NET Framework to write IVI-COM Console Applications in Visual C#.	M924x_IVI_ProgrammingGuide.pdf	PDF
IVI Driver reference (help system)	Provides detailed documentation of the IVI-COM and IVI-C driver API functions, as well as information to help you get started with using the IVI drivers in your application development environment.	AgInfiniiVision.chm	CHM (Microsoft Help Format)
LabVIEW Driver Reference	Provides detailed documentation of the LabVIEW G Driver API functions.	KtM924x_LabVIEW_Help.chm	CHM (Microsoft Help Format)

- To find the latest versions of the user documentation, go to www.keysight.com/manuals/M9241A.

See Also The data sheet introduces the product and provides full product specifications. You can find the data sheet at: www.keysight.com/products/M9241A

The Keysight M9241/42/43A PXIe Oscilloscopes and M9240A AutoProbe Power Module Security Guide is available at www.keysight.com/find/security.

Overall Process Flow

Perform the following steps:

- 1** Write source code using Microsoft Visual Studio 2010 with .NET Visual C# running on Windows 7.
- 2** Compile source code using the .NET Framework Library.
- 3** Produce an Assembly.exe file – this file can run directly from Microsoft Windows without the need for any other programs.
 - When using the Visual Studio Integrated Development Environment (IDE), the Console Applications you write are stored in conceptual containers called **Solutions** and **Projects**.
 - You can view and access Solutions and Projects using the **Solution Explorer** window (**View > Solution Explorer**).

2 Installing Hardware, Software, and Licenses

Perform the following steps:

- 1 Unpack and inspect all hardware.
- 2 Verify the shipment contents.
- 3 Install the software. Note the following order when installing software.
 - a Install Microsoft Visual Studio 2010 with .NET Visual C# running on Windows 7.

You can also use a free version of Visual Studio Express 2010 tools from:
<http://www.microsoft.com/visualstudio/eng/products/visual-studio-2010-express>

The following steps, defined in the Keysight M9241/42/43A PXIe Oscilloscopes and M9240A AutoProbe Power Module Startup Guide, but repeated here must be completed before programmatically controlling the M9241/42/43A PXIe oscilloscope hardware with their IVI drivers.

- b Install Keysight IO Libraries Suite (IOLS), version 17.2 or later; this installation includes Keysight Connection Expert.
- c Install the M9241/42/43A PXIe oscilloscope software version 7.0 or later; this installation includes the AgInfiniiVision IVI driver version 2.2.8.0 or later.
- d Install the M9018A PXIe Chassis driver software, version 1.3.443.1 or later.

Driver software includes all IVI-COM, IVI-C, and LabVIEW G Drivers along with Soft Front Panel (SFP) programs and documentation. All of these items may be downloaded from the Keysight product websites:

- <http://www.keysight.com/find/iosuite> > Select Technical Support > Select the Drivers, Firmware & Software tab > Download the Keysight IO Libraries Suite Recommended
- <http://www.keysight.com/support/M9241A> > Select Technical Support > Select the Drivers, Firmware & Software tab > Download the Instrument Driver.

- <http://www.keysight.com/find/m9018a> > Select Technical Support > Select the Drivers, Firmware & Software tab > Download the Instrument Driver.
 - <http://www.keysight.com/find/ivi> - download other installers for Keysight IVI-COM drivers
- 4 Install the hardware modules and make cable connections.
 - 5 Verify operation of the modules (or the system that the modules create).

NOTE

Before programming or making measurements, conduct a Self-Test on each M924xA PXIe oscilloscope to make sure there are no problems with the modules, cabling, or backplane trigger mapping.

Once the software and hardware are installed, and after Self-Test has been performed, the M924xA PXIe oscilloscopes are ready to be programmatically controlled.

3 APIs for the M9241/42/43A PXIe Oscilloscopes

IVI Compliant or IVI Class Compliant / 13

IVI Driver Types / 14

IVI Driver Hierarchy / 16

Class-Complaint and Instrument-Specific Hierarchies for the M924xA / 17

Naming Conventions Used to Program IVI Drivers / 19

The following IVI driver terminology may be used when describing the Application Programming Interfaces (APIs) for the M9241/42/43A PXIe oscilloscopes.

IVI [Interchangeable Virtual Instruments] - a standard instrument driver model defined by the IVI Foundation that enables engineers to exchange instruments made by different manufacturers without rewriting their code.

www.ivifoundation.org

IVI Instrument Classes (Defined by the IVI Foundation)

Currently, there are 13 IVI Instrument Classes defined by the IVI Foundation. The M9241/42/43A PXIe oscilloscope belongs to the Oscilloscope IVI Instrument Class and are therefore is described as a "Class" module.

- DC Power Supply
- AC Power Supply
- DMM
- Function Generator
- Oscilloscope
- Power Meter
- RF Signal Generator
- Spectrum Analyzer
- Switch
- Upconverter
- Downconverter
- Digitizer

3 APIs for the M9241/42/43A PXIe Oscilloscopes

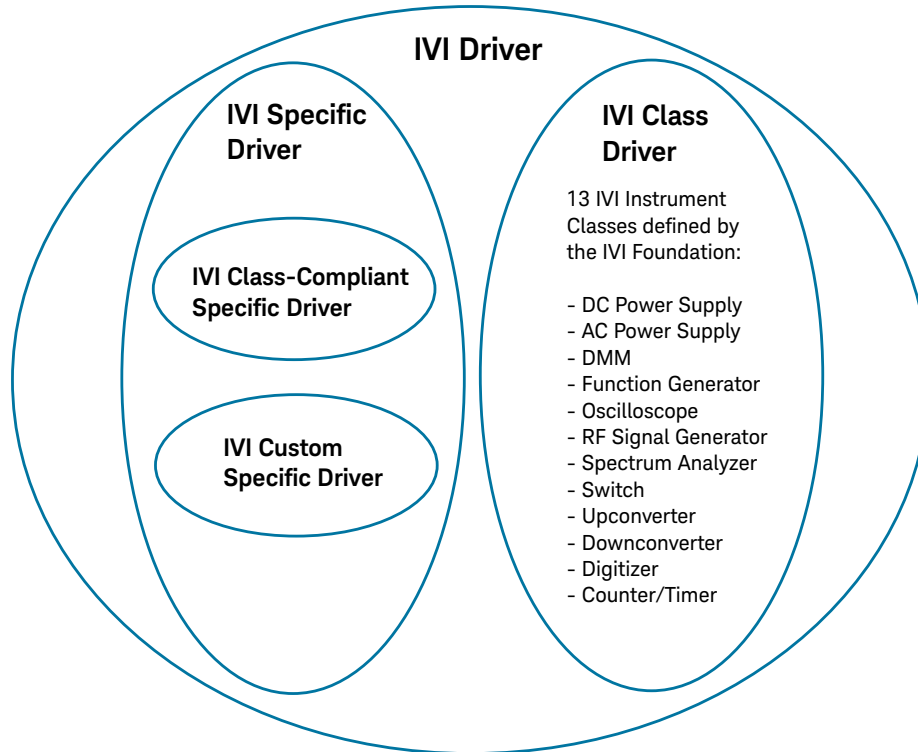
- Counter/Timer

IVI Compliant or IVI Class Compliant

The M9241/42/43A PXIe oscilloscopes driver is IVI Compliant and IVI Class Compliant because it belongs to one of the 13 IVI Instrument Classes defined by the IVI Foundation.

- IVI Compliant – means that the IVI driver follows architectural specifications for these categories:
 - Installation
 - Inherent Capabilities
 - Cross Class Capabilities
 - Style
 - Custom Instrument API
- IVI Class Compliant – means that the IVI driver implements one of the 13 IVI Instrument Classes
 - If an instrument is IVI Class Compliant, it is also IVI Compliant
 - Provides one of the 13 IVI Instrument Class APIs in addition to a Custom API
 - Custom API may be omitted (unusual)
 - Simplifies exchanging instruments

IVI Driver Types



- IVI Driver
 - Implements the Inherent Capabilities Specification
 - Complies with all of the architecture specifications
 - May or may not comply with one of the 13 IVI Instrument Classes
 - Is either an IVI Specific Driver or an IVI Class Driver
- IVI Class Driver
 - Is an IVI Driver needed only for interchangeability in IVI-C environments
 - The IVI Class may be IVI-defined or customer-defined
- IVI Specific Driver
 - Is an IVI Driver that is written for a particular instrument such as the M9241/42/43A PXIe oscilloscopes
 - 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

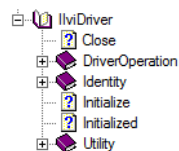
- Is an IVI Specific Driver that is not compliant with any one of the 13 IVI defined class specifications
- Not interchangeable

IVI Driver Hierarchy

When writing programs, you will be using the interfaces (APIs) available to the IVI-COM driver.

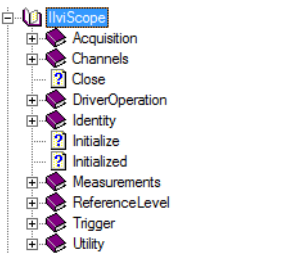
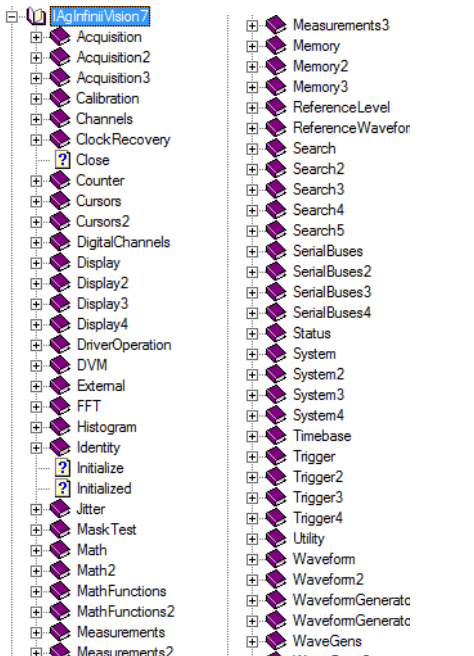
- The core of every IVI-COM driver is a single object with many interfaces.
- These interfaces are organized into two hierarchies: Class-Compliant Hierarchy and Instrument-Specific Hierarchy – and both include the IIVI driver interfaces.
 - Class-Compliant Hierarchy - Because the M9241/42/43A PXIe oscilloscopes belong to one of the 13 IVI Classes, there is a Class-Compliant Hierarchy in its IVI Driver.
 - The M9241/42/43A PXIe oscilloscope's class-compliant hierarchy has IIVI_Scope at the root (where IIVI_Scope is the driver name).
 - IIVI_Scope is the root interface and contains references to child interfaces, which in turn contain references to other child interfaces. Collectively, these interfaces define the Class-Compliant Hierarchy.
 - Instrument-Specific Hierarchy
 - The M9241/42/43A PXIe oscilloscope's instrument-specific hierarchy has IIVI_AgInfiniiVision7 at the root (where AgInfiniiVision is the driver name).
 - IIVI_AgInfiniiVision7 is the root interface and contains references to child interfaces, which in turn contain references to other child interfaces. Collectively, these interfaces define the Instrument-Specific Hierarchy.
- The IIVI driver interfaces are incorporated into both hierarchies: Class-Compliant Hierarchy and Instrument-Specific Hierarchy.

The IIVI driver is the root interface for IVI Inherent Capabilities which are what the IVI Foundation has established as a set of functions and attributes that all IVI drivers must include – irrespective of which IVI instrument class the driver supports. These common functions and attributes are called IVI inherent capabilities and they are documented in IVI-3.2 – Inherent Capabilities Specification. Drivers that do not support any IVI instrument class must still include these IVI inherent capabilities.



Class-Compliant and Instrument-Specific Hierarchies for the M924xA

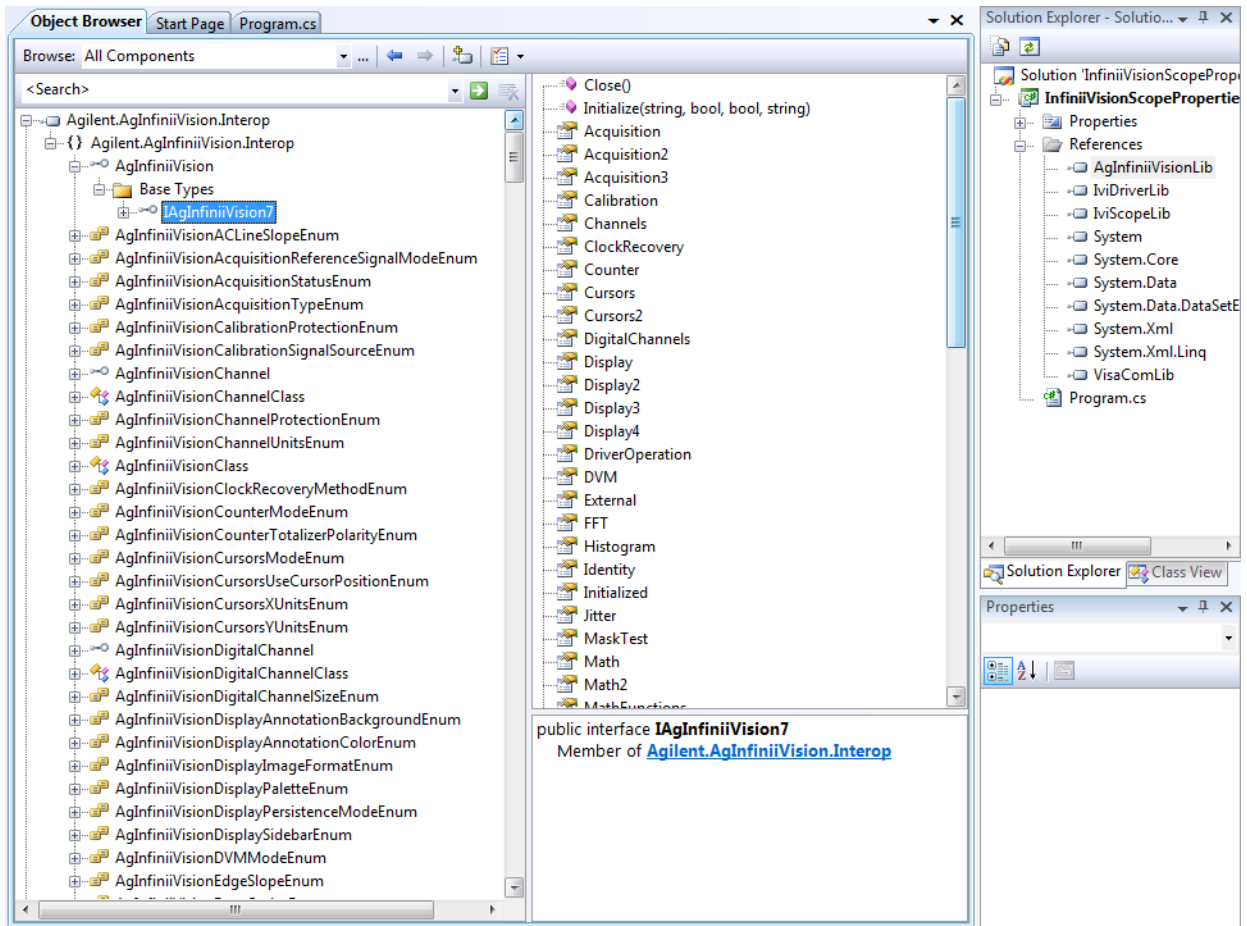
The following table lists the class-compliant and instrument-specific hierarchy interfaces for the M9241/42/43A PXIe oscilloscopes.

IVI Class-Compliant Hierarchy IviScope is the driver name IviScope is the root interface	Keysight M924xA PXIe Oscilloscope Instrument-Specific Hierarchy AgInfiniiVision is the driver name IAgInfiniiVision7 is the root interface
 <ul style="list-style-type: none"> [-] IviScope <ul style="list-style-type: none"> [-] Acquisition [-] Channels [-] Close [-] DriverOperation [-] Identity [-] Initialize [-] Initialized [-] Measurements [-] ReferenceLevel [-] Trigger [-] Utility 	 <ul style="list-style-type: none"> [-] IAgInfiniiVision7 <ul style="list-style-type: none"> [-] Acquisition [-] Acquisition2 [-] Acquisition3 [-] Calibration [-] Channels [-] ClockRecovery [-] Close [-] Counter [-] Cursors [-] Cursors2 [-] DigitalChannels [-] Display [-] Display2 [-] Display3 [-] Display4 [-] DriverOperation [-] DVM [-] External [-] FFT [-] Histogram [-] Identity [-] Initialize [-] Initialized [-] Jitter [-] Mask Test [-] Math [-] Math2 [-] MathFunctions [-] MathFunctions2 [-] Measurements [-] Measurements2 [-] Measurements3 [-] Memory [-] Memory2 [-] Memory3 [-] ReferenceLevel [-] ReferenceWaveform [-] Search [-] Search2 [-] Search3 [-] Search4 [-] Search5 [-] SerialBuses [-] SerialBuses2 [-] SerialBuses3 [-] SerialBuses4 [-] Status [-] System [-] System2 [-] System3 [-] System4 [-] Timebase [-] Trigger [-] Trigger2 [-] Trigger3 [-] Trigger4 [-] Utility [-] Waveform [-] Waveform2 [-] WaveformGenerators [-] WaveformGenerators2 [-] WaveGens [-] WaveGens2

When Using Visual Studio

To view the interfaces available in the M924xA PXIe oscilloscope driver, right-click AgInfiniiVisionLib library file, in the References folder, from the Solution Explorer window and select View in Object Browser.

3 APIs for the M9241/42/43A PXIe Oscilloscopes



Naming Conventions Used to Program IVI Drivers

General IVI Naming Conventions

- All instrument class names start with "Ivi"
 - Example: IviScope, IviDmm
- Function names
 - One or more words use PascalCasing
 - First word should be a verb

IVI-COM Naming Conventions

- Interface naming
 - Class compliant: Starts with "IIVI"
 - <ClassName>
 - Example: IIviScope, IIviDmm
- 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 IIVI-COM
 - <ClassName><descriptive words>Enum
 - Example: IviScopeTriggerCouplingEnum
- Enum values do not end in "Enum" but use the last word to differentiate
 - Examples: IviScopeTriggerCouplingAC and IviScopeTriggerCouplingDC

3 APIs for the M9241/42/43A PXIe Oscilloscopes

4 Creating a Project with IVI-COM Using C-Sharp

Step 1 - Create a Console Application / 22

Step 2 - Add References / 23

Step 3 - Add "using" Statements / 25

Step 4 - Create Instance of the IVI-COM Driver / 26

Step 5 - Initialize the Driver Instance / 27

Step 6 - Write the Program Steps / 33

Step 7 - Close the Driver / 34

Step 8 - Building and Running a Complete Program Using Visual C-Sharp / 35

Additional Example Programs / 39

This tutorial walks through the various steps required to create a console application using Visual Studio and C#. It demonstrates how to instantiate a driver instance, set the resource name and various initialization values, initialize the driver instance, print various driver properties to a console, check drivers for errors, report errors if they occur, and close the driver.

At the end of this tutorial is a complete example program that shows what the console application looks like if you follow all of these steps.

Step 1 - Create a Console Application

NOTE

Projects that use a Console Application do not show a Graphical User Interface (GUI) display.

- 1 Launch Visual Studio and create a new Console Application in Visual C# by selecting: **File > New > Project** and select a Visual C#, Windows, **Console Application**.
- 2 Enter "InfiniiVisionScopeProperties" as the Name of the project and click OK.

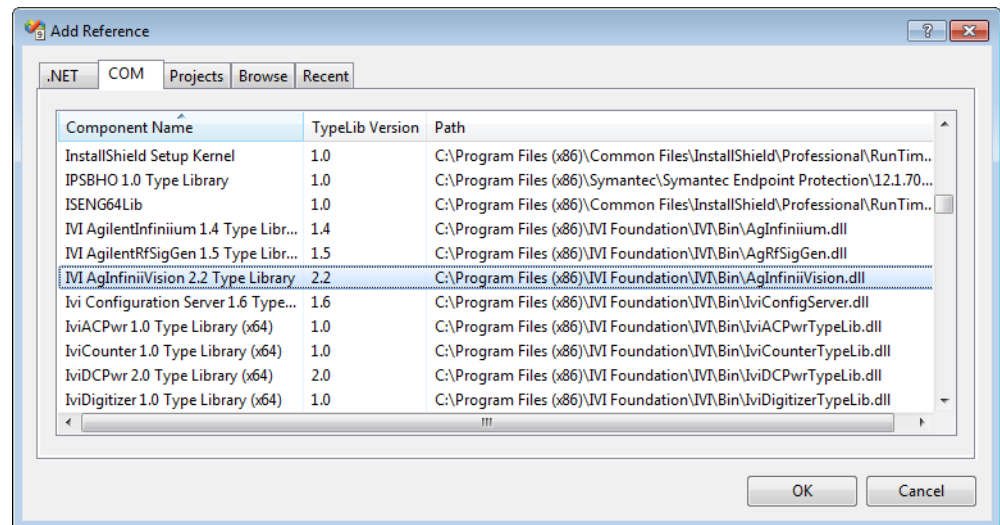
NOTE

When you select New, Visual Studio will create an empty "Program.cs" file that includes some necessary code, including using statements. This code is required, so do not delete it.

Step 2 - Add References

In order to access the M924xA PXIe oscilloscope driver interfaces, references to their drivers (DLL) must be created.

- 1 For this step, Solution Explorer must be visible (**View > Solution Explorer**) and the "Program.cs" editor window must be visible; select the Program.cs tab to bring it to the front view.
- 2 In Solution Explorer, right-click on **References** and select **Add Reference...**
- 3 From the Add Reference dialog box, select the **COM** tab.
- 4 Click on any of the type libraries under the "Component Name" heading and enter the letter "I". (All IVI drivers begin with IVI so this will move down the list of type libraries that begin with "I".)



NOTE

If you have not installed the IVI driver for the M924xA PXIe oscilloscope products (as listed in **Chapter 2**, "Installing Hardware, Software, and Licenses," starting on page 9), their IVI drivers will not appear in this list.

Also, the TypeLib Version that appears will depend on the version of the IVI driver that is installed. The version numbers change over time and typically increase as new drivers are released.

If the TypeLib Version that is displayed on your system is higher than the ones shown in this example, your system simply has newer versions – newer versions may have additional commands available.

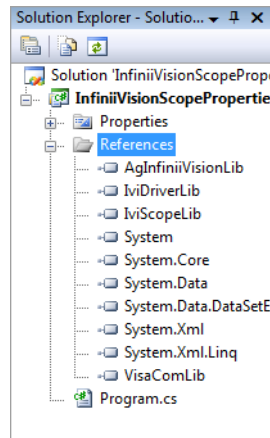
To get the IVI drivers to appear in this list, you must close this Add Reference dialog, install the IVI drivers, and come back to this section and repeat **"Step 2 - Add References"** on page 23.

- 5 Scroll to IVI section and select the following type library. Then click **OK**.
 - IVI AgInfiniiVision 2.2 Type Library

NOTE

When any of the references for the AgInfiniiVision are added, the IviDriver 1.0 Type Library is also automatically added. This is visible as IviDriverLib under the project Reference; this reference houses the interface definitions for IVI inherent capabilities which are located in the file IviDriverTypeLib.dll (dynamically linked library).

- 6 These selected type libraries appear under the **References** node, in Solution Explorer, as:



NOTE

The program looks same as before you added the References, with the difference that the IVI drivers that are referenced are now available for use.

To allow your program to access the IVI drivers without specifying full path names of each interface or enum, you need to add using statements to your program.

Step 3 - Add "using" Statements

All data types (interfaces and enums) are contained within namespaces. (A namespace is a hierarchical naming scheme for grouping types into logical categories of related functionality. Design tools, such as Visual Studio, can use namespaces which makes it easier to browse and reference types in your code.) The C# "using" statement allows the type name to be used directly. Without the "using" statement, the complete namespace-qualified name must be used.

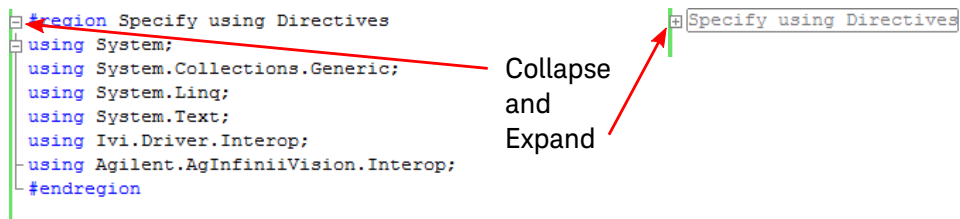
To allow your program to access the IVI driver without having to type the full path of each interface or enum, type the following using statements immediately below the other using statements.

These using statements should be added to your program:

```
using Ivi.Driver.Interop;
using Agilent.AgInfiniiVision.Interop;
```

NOTE

You can create sections of code in your program that can be expanded and collapsed by surrounding the code with #region and #endregion preprocessing directives. Select - or + symbol to collapse or expand the region.



```
#region Specify using Directives
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Ivi.Driver.Interop;
using Agilent.AgInfiniiVision.Interop;
#endregion
```

Collapse and Expand

Step 4 - Create Instance of the IVI-COM Driver

There are two ways to instantiate (create an instance of) the IVI-COM drivers:

- COM Session Factory
- Direct Instantiation

Because the InfiniiVision oscilloscopes are considered Class modules (because they belong to one of the 13 IVI Classes), the COM Session Factory can be used to create instances of their IVI-COM drivers.

```
// Create an instance of the session factory
IIviSessionFactory factory = new IviSessionFactoryClass();

// Ask the session factory to create an instance of
// the appropriate driver based on a logical name
IIviScope iviscope = (IIviScope)factory.CreateDriver("MyLogicalName");
```

When direct instantiation of the InfiniiVision oscilloscope driver is used:

```
// Instantiate the driver class directly
AgInfiniiVision driver = new AgInfiniiVision();
```

The IVI-COM drivers may not be interchangeable with other oscilloscope modules.

The `new` operator is used in C# to create an instance of the driver.

```
// Create driver instance
AgInfiniiVision driver = new AgInfiniiVision();
```

Step 5 - Initialize the Driver Instance

The `Initialize()` method is required when using any IVI driver. It establishes a communication link (an "I/O session") with an instrument and it must be called before the program can do anything with an instrument or work in simulation mode.

The `Initialize()` method has a number of options that can be defined. In this example, we prepare the `Initialize()` method by defining only a few of the parameters, then we call the `Initialize()` method with these parameters:

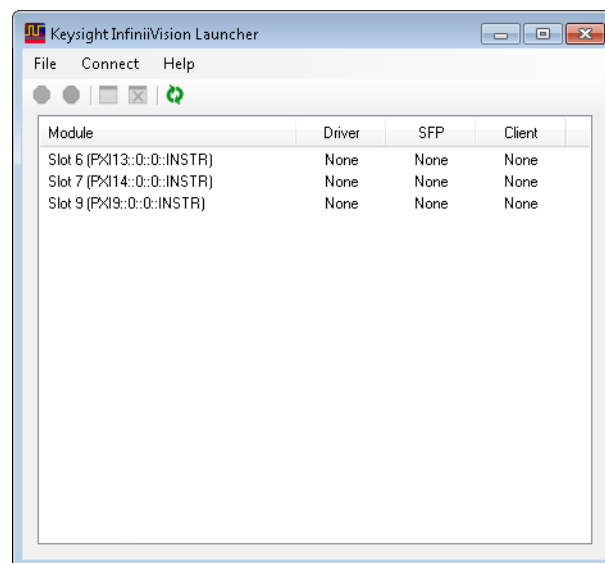
Resource Names

- If you are using Simulate Mode, you can set the Resource Name address string to:

```
string resourceDesc = "%";
```

- If you are actually establishing a communication link (an "I/O session") with an instrument, you need to determine the Resource Name address string (VISA address string) that is needed. You can use an IO application such as Agilent/Keysight Connection Expert, Agilent/Keysight Command Expert, National Instruments Measurement and Automation Explorer (MAX), or you can use the Keysight product's Soft Front Panel (SFP) to get the physical Resource Name string.

Using the Keysight M924x InfiniiVision SFP, you might get the following Resource Name address strings.



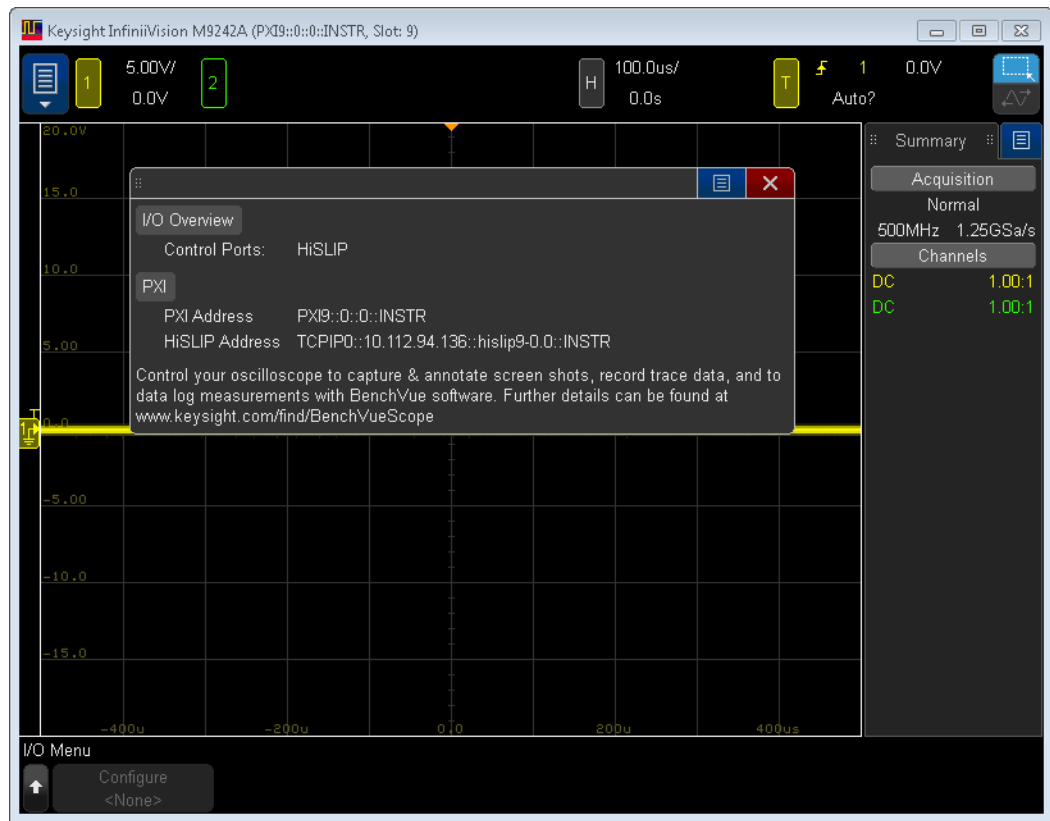
Module Name	M9241A PXIe Oscilloscope	M9243A PXIe Oscilloscope	M9242A PXIe Oscilloscope
Slot Number	6	7	9
VISA Address	PXI13::0::0::INSTR	PXI14::0::0::INSTR	PXI9::0::0::INSTR

When running the remote program on the PXIe chassis controller PC, you could, for example, use the following resource string.

```
string resourceDesc = "PXI9::0::0::INSTR";
```

When running the remote program on some other controller PC on the network, you need to find the "HiSLIP Address" Resource Name address string. To do this:

- a In the the Keysight M924x InfiniiVision SFP, select one of the InfiniiVision modules and click its **Show Front Panel** icon.
- b In the oscilloscope module's Front Panel graphical user interface, choose **(Menu) > Utilities > I/O Menu**.
- c In the dialog box that appears, take note of the HiSLIP address.



So, when running the remote program on a controller PC other than the PXIe chassis controller, you could, for example, use the following resource string.

```
string resourceDesc = "TCPIP::10.112.94.136::hislip9-0.0::INSTR";
```

You could also include the chassis controller host name in place of the IP address (10.112.94.136) in the "HiSLIP Address" string.

NOTE

In order to control a M924x InfiniiVision oscilloscope module, its Front Panel interface must be running.

This resource string is similar to one you would use for a standalone InfiniiVision oscilloscope.

```
string resourceDesc = "TCPIP0::141.121.230.115::hislip0::INSTR";
```

PXI and HiSLIP Address Differences

The PXI interface does not support sending the <END> message and depends on a new line <NL> to terminate a command or query (similar to a telnet socket port on a benchtop oscilloscope). You may run into this issue when using raw binary writes over the PXI interface. For example:

```
AgInfiniiVision driver = new AgInfiniiVision();

string initOptions = "QueryInstrStatus=true, Simulate=false,
    DriverSetup= Model=, Trace=false, TraceName=c:\\temp\\traceOut";
bool idquery = true;
bool reset = false;

string pxiAddr = "PXI35::0::0::INSTR";
string hislipAddr = "TCPIP0::localhost::hislip35-0.0::INSTR";

driver.Initialize(pxiAddr, idquery, reset, initOptions);
...
...
...
byte[] cmd = Encoding.ASCII.GetBytes("*IDN?\n");
driver.System4.DirectIO.IO.Write(cmd, cmd.Length);
```

When using a HiSLIP address, the "\n" is not required. However, when using the PXI address, the "\n" is required.

Initialize() Parameters

NOTE

Although the `Initialize()` method has a number of options that can be defined (see [Initialize Options](#) below), we are showing this example with a minimum set of options to help minimize complexity.

```
// Define driver Initialize options
string initOptions = "QueryInstrStatus=true, Simulate=false, DriverSetup
=
```

```

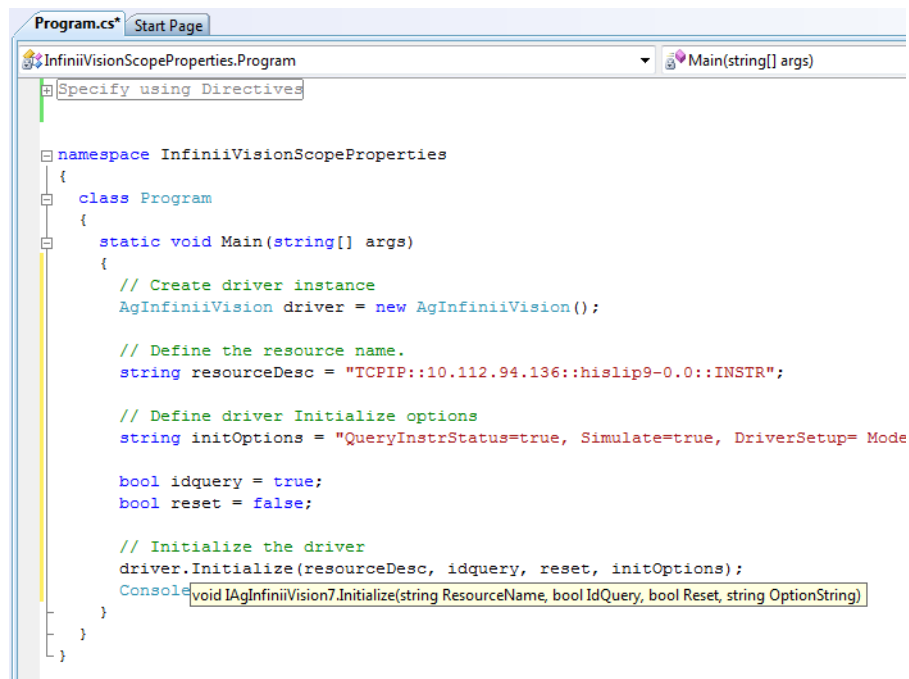
        Model=, Trace=false, TraceName=c:\\temp\\traceOut";

bool idquery = true;
bool reset = false;

// Initialize the driver.
driver.Initialize(resourceDesc, idquery, reset, initOptions);
Console.WriteLine("Driver Initialized");

```

The following picture shows how IntelliSense is invoked by simply rolling the cursor over the word "Initialize".



NOTE

One of the key advantages of using C# in the Microsoft Visual Studio Integrated Development Environment (IDE) is IntelliSense. IntelliSense is a form of auto-completion for variable names and functions and a convenient way to access parameter lists and ensure correct syntax. This feature also enhances software development by reducing the amount of keyboard input required.

Initialize() Options

The following table describes options that are most commonly used with the `Initialize()` method.

Property Type and Example Value	Description of Property
<pre>string ResourceName = "PXI[bus]::device[::function][::INSTR]"; string ResourceName = "PXI13::0::0::INSTR;PXI14::0::0::INSTR;PXI15::0:: 0::INSTR;PXI16::0::0::INSTR";</pre>	<p>ResourceName – The driver is typically initialized using a physical resource name descriptor, often a VISA resource descriptor.</p> <p>See the procedure in the "Resource Names" on page 27 section.</p>
<pre>bool IdQuery = true;</pre>	<p>Setting the ID query to false prevents the driver from verifying that the connected instrument is the one the driver was written for.</p> <p>If IdQuery is set to true, this will query the instrument model and fail initialization if the model is not supported by the driver.</p>
<pre>bool Reset = true;</pre>	<p>Setting Reset to true instructs the driver to initially reset the instrument.</p>

Property Type and Example Value	Description of Property
<pre>string OptionString = "QueryInstrStatus=true, Simulate=true, DriverSetup= Trace=false";</pre>	<p>OptionString - Setup the following initialization options:</p> <ul style="list-style-type: none"> ▪ QueryInstrStatus=true (Specifies whether the IVI specific driver queries the instrument status at the end of each user operation.) ▪ Simulate=true (Setting Simulate to true instructs the driver to not attempt to connect to a physical instrument, but use a simulation of the instrument instead.) ▪ Cache=false (Specifies whether or not to cache the value of properties.) ▪ InterchangeCheck=false (Specifies whether the IVI specific driver performs interchangeability checking.) ▪ RangeCheck=false (Specifies whether the IVI specific driver validates attribute values and function parameters.) ▪ RecordCoercions=false (Specifies whether the IVI specific driver keeps a list of the value coercions it makes for ViInt32 and ViReal64 attributes.)
<pre>DriverSetup= Trace=false";</pre>	<ul style="list-style-type: none"> ▪ DriverSetup= (This is used to specify settings that are supported by the driver, but not defined by IVI. If the Options String parameter (OptionString in this example) contains an assignment for the Driver Setup attribute, the Initialize function assumes that everything following 'DriverSetup=' is part of the assignment.) ▪ Model= (Instrument model to use during simulation) ▪ Trace=false (If false, an output trace log of all driver calls is not saved in an XML file.)

If these drivers were installed, additional information can be found under Initializing the IVI-COM Driver from the following:

AgInfiniiVision IVI
Driver Reference

Start > All Programs > Keysight Instrument Drivers > IVI-COM-C AgInfiniiVision 2.2.3 Oscilloscope (Open driver root folder)

In the driver root folder, double-click the "AgInfiniiVision.chm" file to open the driver reference.

Step 6 - Write the Program Steps

At this point, you can add program steps that use the driver instance to perform tasks. For example:

```
// Print a few IIVI driver identity properties
Console.WriteLine("Identifier: {0}", driver.Identity.Identifier);
Console.WriteLine("Revision: {0}", driver.Identity.Revision);
Console.WriteLine("Vendor: {0}", driver.Identity.Vendor);
Console.WriteLine("Description: {0}", driver.Identity.Description);
Console.WriteLine("Model: {0}", driver.Identity.InstrumentModel);
Console.WriteLine("FirmwareRev: {0}", driver.Identity.InstrumentFirmwareRevision);
Console.WriteLine("Serial #: {0}", driver.System.SerialNumber);
Console.WriteLine("\nSimulate: {0}\n", driver.DriverOperation.Simulate);

//Initiate the acquisition and return the acquired waveform data
System.Double[] WaveformArray = { 128, 64, 32, 16, 8, 4, 2, 1 };
System.Double InitialX = 10;
System.Double XIncrement = 5;
driver.Measurements.AutoSetup();

driver.Measurements.Initiate();

driver.Measurements.get_Item("Channel1").FetchWaveform(ref WaveformArray, ref InitialX, ref XIncrement);
Console.WriteLine("\nInitialX: {0} XIncrement: {1}", InitialX, XIncrement);

System.Int32 Points = WaveformArray.Length;
Console.WriteLine("\nData Points: {0}\nWaveformArray: \n", Points);
if (Points > 100) Points = 100;
for (int i = 0; i < Points; i++)
    Console.WriteLine("{0}, ", WaveformArray[i]);
Console.WriteLine();

// Check instrument for errors
int errorNum = -1;
string errorMsg = null;
Console.WriteLine();
while (errorNum != 0)
{
    driver.Utility.ErrorQuery(ref errorNum, ref errorMsg);
    Console.WriteLine("ErrorQuery: {0}, {1}", errorNum, errorMsg);
}
```

Step 7 - Close the Driver

Calling `close()` at the end of the program is required by the IVI specification when using any IVI driver.

CAUTION

Important! `close()` may be the most commonly missed step when using an IVI driver. Failing to do this could mean that system resources are not freed up and your program may behave unexpectedly on subsequent executions.

```
// Close the driver
if (driver != null && driver.Initialized)
{
    driver.Close();
    Console.WriteLine("Driver Closed");
}

// Write "Done" to console
Console.WriteLine("Done - Press Enter to Exit");
Console.ReadLine();
```

Step 8 - Building and Running a Complete Program Using Visual C-Sharp

Build your console application and run it to verify it works properly.

- 1 Open the solution file `SolutionNameThatYouUsed.sln` in Visual Studio 2010.
- 2 Set the appropriate platform target for your project.
 - In many cases, the default platform target (Any CPU) is appropriate.
 - However, if you are using a 64-bit PC (such as Windows 7) to build a .NET application that uses a 32-bit IVI-COM driver, you may need to specify your project's platform target as x86.
- 3 Choose **Project > ProjectNameThatYouUsed Properties** and select **Build | Rebuild Solution**.
 - Tip: You can also do the same thing from the **Debug** menu by clicking **Start Debugging** or pressing the F5 key.

Example Program - Code Structure

After placing the example program statements in try/catch/finally blocks, separating the driver declaration from instance creation, and adding #region and #endregion preprocessor directives, you end up with an example program structure that looks like:

```

Program.cs Start Page
InfiniiVisionScopeProperties.Program Main(string[] args)
#region Specify using Directives
namespace InfiniiVisionScopeProperties
{
    class Program
    {
        static void Main(string[] args)
        {
            AgInfiniiVision driver = null;
            try
            {
                // Create driver instance
                driver = new AgInfiniiVision();

                Initialize Driver Instances

                Print Driver Properties

                Perform Tasks

                Check for Errors
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
            finally
            {
                Close Driver Instances
            }

            // Write "Done" to console
            Console.WriteLine("Done - Press Enter to Exit");
            Console.ReadLine();
        }
    }
}

```

Example Program - Full Code Listing

The example program full code listing looks like:

```

#region Specify using Directives
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Ivi.Driver.Interop;
using Agilent.AgInfiniiVision.Interop;
#endregion

namespace InfiniiVisionScopeProperties
{
    class Program
    {
        static void Main(string[] args)
        {
            AgInfiniiVision driver = null;
            try

```

```

{
    // Create driver instance
    driver = new AgInfiniiVision();

    #region Initialize Driver Instances
    // Define the resource name.
    string resourceDesc = "TCPIP::10.112.94.136::hislip9-0.0::INSTR"
;

    // Define driver Initialize options
    string initOptions = "QueryInstrStatus=true, Simulate=false, DriverSetup= Model=, Trace=false, TraceName=c:\\temp\\traceOut";

    bool idquery = true;
    bool reset = false;

    // Initialize the driver
    driver.Initialize(resourceDesc, idquery, reset, initOptions);
    Console.WriteLine("Driver Initialized");
    #endregion

    #region Print Driver Properties
    // Print a few IIviDriverIdentity properties
    Console.WriteLine("Identifier: {0}", driver.Identity.Identifier
);
    Console.WriteLine("Revision: {0}", driver.Identity.Revision);
    Console.WriteLine("Vendor: {0}", driver.Identity.Vendor);
    Console.WriteLine("Description: {0}", driver.Identity.Descriptio
n);
    Console.WriteLine("Model: {0}", driver.Identity.Instrument
Model);
    Console.WriteLine("FirmwareRev: {0}", driver.Identity.Instrument
FirmwareRevision);
    Console.WriteLine("Serial #: {0}", driver.System.SerialNumber
);
    Console.WriteLine("\nSimulate: {0}\
n", driver.DriverOperation.Simulate);
    #endregion

    #region Perform Tasks
    //Initiate the acquisition and return the acquired waveform data
    System.Double[] WaveformArray = { 128, 64, 32, 16, 8, 4, 2, 1 };
    System.Double InitialX = 10;
    System.Double XIncrement = 5;
    driver.Measurements.AutoSetup();

    driver.Measurements.Initiate();

    driver.Measurements.get_Item("Channel1").FetchWaveform(ref Wavef
ormArray, ref InitialX, ref XIncrement);
    Console.WriteLine("\
nInitialX: {0} XIncrement: {1}", InitialX, XIncrement);

    System.Int32 Points = WaveformArray.Length;
    Console.Write("\n\nData Points: {0}\n\nWaveformArray: \
n", Points);
    if (Points > 100) Points = 100;

```

```

        for (int i = 0; i < Points; i++)
            Console.WriteLine("{0}, ", WaveformArray[i]);
        Console.WriteLine();
    #endregion

    #region Check for Errors
    // Check instrument for errors
    int errorNum = -1;
    string errorMsg = null;
    Console.WriteLine();
    while (errorNum != 0)
    {
        driver.Utility.ErrorQuery(ref errorNum, ref errorMsg);
        Console.WriteLine("ErrorQuery: {0}, {1}", errorNum, errorMsg);
    }
    #endregion
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
finally
{
    #region Close Driver Instances
    // Close the driver
    if (driver != null && driver.Initialized)
    {
        driver.Close();
        Console.WriteLine("Driver Closed");
    }
    #endregion
}

// Write "Done" to console
Console.WriteLine("Done - Press Enter to Exit");
Console.ReadLine();
}
}
}

```

Additional Example Programs

Additional example programs can be found in: C:\Program Files (x86)\IVI Foundation\IVI\Drivers\AgInfiniiVision\Examples

5 Creating a Project with IVI-COM Using Python

You can use the Python programming language with the "comtypes" package to control Keysight oscilloscopes using the IVI-COM library.

The Python language and "comtypes" package can be downloaded from the web at <http://www.python.org/> and <http://starship.python.net/crew/theller/comtypes/>, respectively.

To run this example with Python and "comtypes":

- 1 Cut-and-paste the code that follows into a file named "example.py".
- 2 Edit the program to use the VISA address of your oscilloscope.
- 3 If "python.exe" can be found via your PATH environment variable, open a Command Prompt window; then, change to the folder that contains the "example.py" file, and enter:

```
python example.py

#
# Oscilloscope IVI-COM instrument-specific AgInfiniiVision
# example in Python using "comtypes"
# *****
# This program illustrates a few commonly used programming
# features of your Keysight oscilloscope.
# *****

# Import Python modules.
# -----
import sys
import array

from comtypes.client import GetModule
from comtypes.client import CreateObject

# Run GetModule once to generate comtypes.gen.AgInfiniiVisionLib
if not hasattr(sys, "frozen"):
    GetModule("C:\Program Files (x86)\IVI Foundation\IVI\Bin\
    AgInfiniiVision.dll")
```

```

from comtypes.gen import AgInfiniiVisionLib

# Global variables (booleans: 0 = False, 1 = True).
# -----

# =====
# Initialize:
# =====
def initialize():

    # Initialize.
    if not my_scope.Initialized:
        my_scope.Initialize(
            'TCPIP0::lab-pxi-5.cos.is.keysight.com::hislip9-0.0::INSTR',
            False,
            False,
            ''
        )

    # Clear the interface.
    my_scope.System2.ClearIO
    print "Interface cleared."

    # Set the Timeout to 15 seconds.
    my_scope.System.TimeoutMilliseconds = 15000 # 15 seconds.
    print "Timeout set to 15000 milliseconds."

    # Query instrument status after each I/O.
    my_scope.DriverOperation.QueryInstrumentStatus = True

    # Get and display the oscilloscope's identity.
    print "Description: '%s'" % my_scope.Identity.Description
    print "Group capabilities: '%s'" % my_scope.Identity.GroupCapabilities
    print "Identifier: '%s'" % my_scope.Identity.Identifier
    print "Instrument firmware revision: '%s'" % my_scope.Identity.InstrumentFirmwareRevision
    print "Instrument manufacturer: '%s'" % my_scope.Identity.InstrumentManufacturer
    print "Instrument model: '%s'" % my_scope.Identity.InstrumentModel
    print "Revision: '%s'" % my_scope.Identity.Revision
    print "Specification major version: '%s'" % my_scope.Identity.SpecificationMajorVersion
    print "Specification minor version: '%s'" % my_scope.Identity.SpecificationMinorVersion
    print "Supported instrument models: '%s'" % my_scope.Identity.SupportedInstrumentModels
    print "Vendor: '%s'" % my_scope.Identity.Vendor
    print "Serial number: '%s'" % my_scope.System.SerialNumber

    # Place oscilloscope in known state.
    my_scope.Utility.Reset()

# =====

```

```

# Capture:
# =====
def capture():

    # Use auto-scale to automatically set up oscilloscope.
    print "Autoscale."
    my_scope.Measurements.AutoSetup()

    # Set trigger mode.
    my_scope.Trigger.Configure(
        Type=AgInfiniiVisionLib.AgInfiniiVisionTriggerTypeEdge,
        Holdoff=0.00000040
    )
    print "Trigger type: %s" % my_scope.Trigger.Type

    # Set edge trigger parameters.
    my_scope.Trigger.Edge.Configure(
        Source='Channell1',
        Level=1.5,
        Slope=AgInfiniiVisionLib.AgInfiniiVisionTriggerSlopePositive
    )
    print "Trigger edge source: %s" % my_scope.Trigger.Source
    print "Trigger edge level: %s" % my_scope.Trigger.Level
    print "Trigger edge slope: %s" % my_scope.Trigger.Edge.Slope

    # Save oscilloscope setup.
    setup_bytes = my_scope.System.GetState()
    # nLength = len(setup_bytes)
    nLength = len(setup_bytes) + 1 # Plus newline character.
    f = open("setup.stp", "wb")
    f.write(bytearray(setup_bytes))
    f.write("\n") # Needs newline or will get error on restore.
    f.close()
    print "Setup bytes saved: %d" % nLength
    print "Setup saved to setup.stp."
    check_instrument_errors()

    # Change oscilloscope settings with individual commands:

    # Set channel 1 vertical scale and offset.
    ch1 = my_scope.Channels.Item('Channell1')
    ch1.Configure(
        Range=4.0,
        Offset=2.075,
        Coupling=AgInfiniiVisionLib.AgInfiniiVisionVerticalCouplingDC,
        ProbeAttenuation=10.0,
        Enabled=True
    )
    ch1.Offset = 2.075 # Parameters can be configured using properties
    print "Channel 1 vertical range: %f" % ch1.Range
    print "Channel 1 vertical offset: %f" % ch1.Offset

    # Set horizontal scale and offset.
    my_scope.Acquisition.ConfigureRecord(
        TimePerRecord=0.005,
        NumberOfPointsMin=1000,
        StartTime=-0.0025

```

```

)
print "Time per record: %f" % my_scope.Acquisition.TimePerRecord
print "Acquisition start time: %f" % my_scope.Acquisition.StartTime
print "Number of points minimum: %d" % my_scope.Acquisition.NumberOfPointsMin

# Set the acquisition type.
my_scope.Acquisition.Type = AgInfiniiVisionLib.AgInfiniiVisionAcquisitionTypeNormal
print "Acquisition type: %s" % my_scope.Acquisition.Type

# Or, configure by loading a previously saved setup.
f = open("setup.stp", "rb")
setup_bytes = f.read()
f.close()
my_scope.System.PutState(array.array('B', setup_bytes))
check_instrument_errors()
print "Setup bytes restored: %d" % len(setup_bytes)

# =====
# Analyze:
# =====
def analyze():

    # Make measurements.
    # -----
    meas_freq = my_scope.Measurements.Item('Channel1').ReadWaveformMeasurement(
        MeasFunction=AgInfiniiVisionLib.AgInfiniiVisionMeasurementFrequency,
        MaxTimeMilliseconds=10000
    )
    print "Measured frequency on channel 1: %f" % meas_freq

    meas_vamp = my_scope.Measurements.Item('Channel1').FetchWaveformMeasurement(
        MeasFunction=AgInfiniiVisionLib.AgInfiniiVisionMeasurementAmplitude
    )
    print "Measured vertical amplitude on channel 1: %f" % meas_vamp

    # Download the screen image.
    # -----
    my_scope.Display2.InvertColorEnabled = False

    image_bytes = my_scope.Display.GetScreenBitmap(
        ImageFormat=AgInfiniiVisionLib.AgInfiniiVisionDisplayImageFormatPNG,
        Palette=AgInfiniiVisionLib.AgInfiniiVisionDisplayPaletteColor
    )
    f = open("screen.png", "wb")
    f.write(bytearray(image_bytes))
    f.close()
    print "Screen image written to screen.png."

    # Download waveform data.
    # -----
    (wfm_data, x_origin, x_increment) = my_scope.Measurements.Item('Channel1').FetchWaveform()

```

```

# print "Waveform array: ", wfm_data
print "Initial X time origin: ", x_origin
print "X increment: ", x_increment

# Get the number of waveform points available.
print "Waveform points available: %s" % my_scope.Acquisition.RecordLength

# Get the waveform data.
wfm_points = len(wfm_data)
print "Number of waveform data values: %d" % wfm_points

# Open file for output.
strPath = "waveform_data.csv"
f = open(strPath, "w")

# Output waveform data in CSV format.
for i in xrange(0, wfm_points - 1):
    time_val = x_origin + (i * x_increment)
    f.write("%E, %f\n" % (time_val, wfm_data[i]))

# Close output file.
f.close()
print "Waveform data written to %s." % strPath

# =====
# Check for instrument errors:
# =====
def check_instrument_errors():

    errors_found = False
    while True:
        (error_code, error_msg) = my_scope.Utility.ErrorQuery()
        if error_code != 0:
            print "ERROR: %d,\"%s\"" % (error_code, error_msg)
            errors_found = True
        else: # Error code == 0.
            break

    if errors_found:
        print "Exited because of error."
        sys.exit(1)

# =====
# Main program:
# =====

my_scope = CreateObject(
    'AgInfiniiVision.AgInfiniiVision'
)

# Initialize the oscilloscope, capture data, and analyze.
initialize()
capture()

```

5 Creating a Project with IVI-COM Using Python

```
analyze()  
  
# Close the I/O session to the instrument.  
my_scope.Close()  
  
print "End of program"
```

6 References

- [Understanding Drivers and Direct I/O, Application Note 1465-3](#) (Keysight literature part number: 5989-0110EN)
- www.ivifoundation.org

6 References

Glossary

A

ADE (application development environment) – An integrated suite of software development programs. ADEs may include a text editor, compiler, and debugger, as well as other tools used in creating, maintaining, and debugging application programs. Example: Microsoft Visual Studio.

API (application programming interface) – An API is a well-defined set of set of software routines through which application program can access the functions and services provided by an underlying operating system or library. Example: IVI Drivers

C

C# (pronounced "C sharp") – C-like, component-oriented language that eliminates much of the difficulty associated with C/C++.

D

Direct I/O Commands sent directly to an instrument, without the benefit of, or interference from a driver. SCPI Example: SENSE:VOLTage:RANGe:AUTO Driver (or device driver) – a collection of functions resident on a computer and used to control a peripheral device.

DLL (dynamic link library) – An executable program or data file bound to an application program and loaded only when needed, thereby reducing memory requirements. The functions or data in a DLL can be simultaneously shared by several applications.

I

Input/Output (I/O) layer The software that collects data from and issues commands to peripheral devices. The VISA function library is an example of an I/O layer that allows application programs and drivers to access peripheral instrumentation.

IVI (Interchangeable Virtual Instruments) – a standard instrument driver model defined by the IVI Foundation that enables engineers to exchange instruments made by different manufacturers without rewriting their code.
www.ivifoundation.org

IVI COM drivers (also known as IVI Component drivers) – IVI COM presents the IVI driver as a COM object in Visual Basic. You get all the intelligence and all the benefits of the development environment because IVI COM does things in a smart way and presents an easier, more consistent way to send commands to an instrument. It is similar across multiple instruments.

M

Microsoft COM (Component Object Model) – The concept of software components is analogous to that of hardware components: as long as components present the same interface and perform the same functions, they are interchangeable. Software components are the natural extension of DLLs. Microsoft developed the COM standard to allow software manufacturers to create new software components that can be used with an existing application program, without requiring that the application be rebuilt. It is this capability that allows T&M instruments and their COM-based IVIComponent drivers to be interchanged.

N

.NET Framework The .NET Framework is an object-oriented API that simplifies application development in a Windows environment. The .NET Framework has two main components: the common language runtime and the .NET Framework class library.

V

VISA (Virtual Instrument Software Architecture) – The VISA standard was created by the VXIplug&play Foundation. Drivers that conform to the VXIplug&play standards always perform I/O through the VISA library. Therefore if you are using Plug and Play drivers, you will need the VISA I/O library. The VISA standard was intended to provide a common set of function calls that are similar across physical interfaces. In practice, VISA libraries tend to be specific to the vendor's interface.

VISA-COM The VISA-COM library is a COM interface for I/O that was developed as a companion to the VISA specification. VISA-COM I/O provides the services of VISA in a COM-based API. VISA-COM includes some higher-level services that are not available in VISA, but in terms of low-level I/O communication capabilities, VISA-COM is a subset of VISA. Agilent VISA-COM is used by its IVIComponent drivers and requires that Agilent VISA also be installed.

Index

Symbols

.NET Framework, 50
#endregion preprocessing directive, 25
#region preprocessing directive, 25

A

ADE (application development environment), 49
AgInfiniiVision driver name, 17
API (application programming interface), 49
APIs for M9241/42/43A PXIe oscilloscopes, 11

B

building a program, 35

C

C#, 49
Cache initialization option, 32
class driver, IVI, 14
class-compliant hierarchy for M924xA, 17
class-compliant specific driver, IVI, 14
class-compliant, IVI, 13
class-compliant, IVI driver hierarchy, 16
code, IVI-COM library example in Python, 41
COM session factory instantiation of IVI-COM driver, 26
compliant, IVI, 13
Console Application, create, 22
copyright, 2
custom specific driver, IVI, 14

D

defined values, 19
Direct I/O, 49
direct instantiation of IVI-COM driver, 26
DLL (dynamic link library), 49
driver instance, initializing, 27
driver types, IVI, 14
driver, closing, 34
driver, IVI, 14

DriverSetup, Initialize() option, 32

E

example program, code structure, 35
END message not supported in PXI interface, 29
enum values, 19
example program, full code listing, 36
example programs, additional, 39

F

function names, 19

G

glossary, 49

H

hierarchy, IVI driver, 16
HiSLIP address, 28
HiSLIP address and PXI address differences, 29

I

IaGInfiniiVision7 root interface, 17
IqQuery, Initialize() option, 31
IIVI driver interface, 16
IIVI scope root interface, 17
Initialize() options, 30
Initialize() parameters, 29
Input/Output (I/O) layer, 50
instrument class names, 19
instrument-specific hierarchy for M924xA, 17
instrument-specific, IVI driver hierarchy, 16
IntelliSense, 30
InterchangeCheck initialization option, 32
interface names, 19
interfaces, viewing in Visual Studio, 17
IVI (Interchangeable Virtual Instruments), 50
IVI class driver, 14
IVI COM drivers, 50

IVI driver, 14
IVI driver naming conventions, 19
IVI driver types, 14
IVI Inherent Capabilities, 16
IVI instrument classes, 11
IVI naming conventions, general, 19
IVI specific driver, 14
IVI-COM driver, creating an instance, 26
IVI-COM example in Python, 41
IVI-COM naming conventions, 19
IviScope driver name, 17

M

Microsoft COM (Component Object Model), 50
Model initialization option, 32

N

naming conventions, IVI drivers, 19
new line required with PXI address message, 29
notices, 2

O

OptionString, Initialize() option, 32

P

program steps, writing, 33
project, creating, 21
project, creating Python, 41
PXI address and HiSLIP address differences, 29
Python, IVI-COM example, 41

Q

QueryInstrStatus initialization option, 32

R

RangeCheck initialization option, 32
RecordCoercions initialization option, 32

Index

reference, IVI driver, [32](#)
references, [47](#)
references, adding, [23](#)
related documentation, [7](#)
Reset, Initialize() option, [31](#)
resource names, [27](#)
ResourceName, Initialize() option, [31](#)
running a program, [35](#)

S

Simulate initialization option, [32](#)
specific driver, IVI, [14](#)
sub-interface names, [19](#)

T

Trace initialization option, [32](#)

U

using statements, adding, [25](#)

V

VISA (Virtual Instrument Software
Architecture), [50](#)
VISA-COM, [51](#)
Visual Studio, viewing interfaces, [17](#)

W

warranty, [2](#)