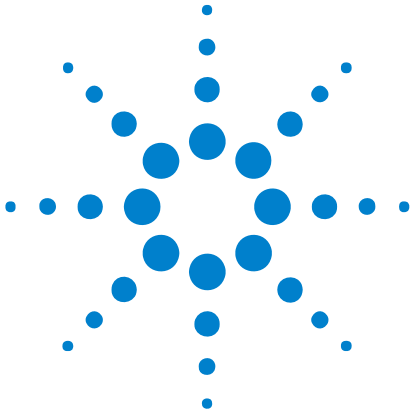


Using the IVI-COM Driver for InfiniiVision Oscilloscopes

Quick Start Guide

Agilent Technologies

Notices

Trademarks

Microsoft®, MS-DOS®, Windows®, Windows 2000®, and Windows XP® are U.S. registered trademarks of Microsoft Corporation.

Adobe®, Acrobat®, and the Acrobat Logo® are trademarks of Adobe Systems Incorporated.

Manual Part Number

54695-97017

Edition

First edition, August 2009

Available in electronic format only

Agilent Technologies, Inc.
1900 Garden of the Gods Road
Colorado Springs, CO 80907 USA

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, Agilent 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. Agilent 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 Agilent 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 Licenses

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.

Restricted Rights Legend

If software is for use in the performance of a U.S. Government prime contract or subcontract, Software is delivered and licensed as "Commercial computer software" as defined in DFAR 252.227-7014 (June 1995), or as a "commercial item" as defined in FAR 2.101(a) or as "Restricted computer software" as defined in FAR 52.227-19 (June 1987) or any equivalent agency regulation or contract clause. Use, duplication or disclosure of Software is subject to Agilent Technologies' standard commercial license terms, and non-DOD Departments and Agencies of the U.S. Government will receive no greater than Restricted Rights as defined in FAR 52.227-19(c)(1-2) (June 1987). U.S. Government users will receive no greater than Limited Rights as defined in FAR 52.227-14 (June 1987) or DFAR 252.227-7015 (b)(2) (November 1995), 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.

IVI-COM Driver for InfiniiVision Oscilloscopes—At a Glance

The IVI Foundation is an open consortium founded in 1998 to promote specifications for programming test instruments. These specifications simplify interchangeability, provide better performance, and reduce the cost of program development and maintenance. For complete information on the IVI Foundation and for the most up-to-date versions of all IVI specifications and components, you can visit the IVI Foundation web site at "www.ivifoundation.org".

The Agilent546XX IVI drivers provides access to the functionality of the Agilent InfiniiVision and 546XX family of oscilloscopes through a COM server or ANSI C API and complies with the IVI specifications. This driver works in any development environment which supports COM or C programming including Microsoft® Visual Basic, Microsoft Visual C++, Microsoft .NET, Agilent VEE Pro, National Instruments LabView, National Instruments LabWindows/CVI, and others.

Supported Instruments

The Agilent 546XX IVI driver currently supports these oscilloscopes:

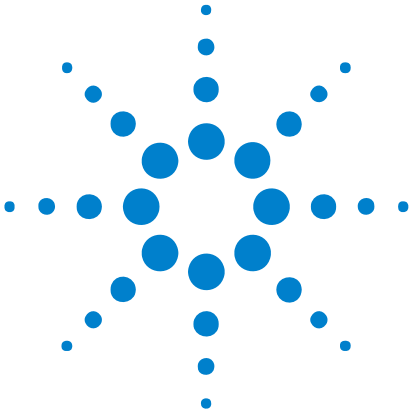
54621A, 54621D, 54622A, 54622D, 54624A, 54641A, 54641D, 54642A, 54642D, DSO5012A, DSO5014A, DSO5032A, DSO5034A, DSO5052A, DSO5054A, DSO6012A, DSO6014A, DSO6014L, DSO6032A, DSO6034A, DSO6034L, DSO6052A, DSO6054A, DSO6054L, DSO6102A, DSO6104A, DSO6104L, DSO7012A, DSO7014A, DSO7032A, DSO7034A, DSO7052A, DSO7054A, DSO7104A, MSO6012A, MSO6014A, MSO6032A, MSO6034A, MSO6052A, MSO6054A, MSO6102A, MSO6104A, MSO7012A, MSO7014A, MSO7032A, MSO7034A, MSO7052A, MSO7054A, MSO7102A, MSO7104A

In This Guide

This guide quickly gets you started using the Agilent546XX IVI-COM driver for Agilent InfiniiVision oscilloscopes. It contains these chapters:

- Chapter 1, "Installing Software," starting on page 7 – describes the software that should be installed on the controller PC.
- Chapter 2, "Performing Additional Set Up," starting on page 13 – describes additional set up steps that should be performed after after software is installed:
  - Adding instrument (oscilloscope) connections using the Agilent Connection Expert.
  - The Agilent Connection Expert comes with the IO Libraries Suite.
  - Adding a logical name for your oscilloscope to the IVI Configuration Store using the Agilent Instrument Explorer.
  - The Agilent Instrument Explorer comes with Agilent T&M Toolkit 2.1 Runtime or Agilent VEE Pro.
  - Adding a logical name to the IVI Configuration Store is necessary for instrument interchangeability.
- Chapter 3, "Quick Start Examples," starting on page 23 – contains short examples of using the Agilent546XX IVI-COM driver in various programming environments along with quick instructions for setting up the environment, building, and running the programs.
- Chapter 4, "Agilent 546XX IVI-COM Driver Notes," starting on page 65 – contains some notes about the online Agilent546XX IVI driver documentation, describes class-compliant driver features that are not available with the Agilent InfiniiVision oscilloscopes, and describes the additional features that are available when using the instrument-specific driver interface.
- Chapter 5, "For More Information," starting on page 71 – points you to the Readme and online Documentation that comes with the Agilent546XX IVI drivers, as well as additional documentation on the IVI Foundation web site.

Contents



1 Installing Software

Controller PC Requirements

Before you can install and use the IVI-COM driver, the contoller PC must have these requirements:

- Supported Operating System:
  - Windows 2000.
  - Windows XP.
  - Windows Vista (32 bit).
- IO Libraries: see “Step 1. Install the Agilent IO Libraries Suite" on page 9.
- IVI Shared Components: see “Step 2. Install the IVI Shared Components" on page 10.

The IVI-COM driver installation checks for these requirements. If not found, the installer either aborts, warns, or installs the required component as appropriate.

Step 1. Install the Agilent IO Libraries Suite

The controller PC must have the Agilent I/O libraries version 15.0 or greater installed.

1 Get the Agilent IO Libraries Suite software.

  The Agilent IO Libraries Suite software is shipped with your oscilloscope on the **Automation-Ready CD**, or you can download the IO Libraries from the Agilent web site at: "www.agilent.com/find/iolib"

2 Install the software from the Automation-Ready CD or downloaded install package.

After installing the IO Libraries Suite, you can make a connection to your oscilloscope and test it. See “Adding Your Oscilloscope Using Agilent Connection Expert" on page 14.

Step 2. Install the IVI Shared Components

The controller PC must have the IVI Shared Components version 1.5.0.0 or later installed.

1 You can download the IVI Shared Components installer from the IVI Foundation web site at: "www.ivifoundation.org/Downloads/SharedComponents.htm"

2 Install the IVI Shared Components by opening or running the downloaded installer file.

Step 3. Install the Agilent546XX IVI-C/IVI-COM Driver

Get the InfiniiVision 5000, 6000, 7000, 546xx Series Oscilloscopes IVI Instrument Drivers from the Agilent web site:

1 Open a web browser to the Agilent web site at: "www.agilent.com/find/ivi-com"

2 Click the **5000, 6000, 7000, 546xx Series Oscilloscopes IVI Instrument Drivers** link.

3 On the 5000, 6000, 7000, 546xx Series Oscilloscopes IVI Instrument Drivers page, download the latest driver and its readme file.

4 Install the downloaded install package.

Step 4. Install the Agilent T&M Toolkit 2.1 Runtime (optional)

The Agilent T&M Toolkit 2.1 Runtime contains the Agilent Instrument Explorer that can be used to create a logical name for your oscilloscope in the IVI Configuration Store.

If you have Agilent VEE installed, the Agilent T&M Toolkit and the Agilent Instrument Explorer are already installed, and the Agilent T&M Toolkit 2.1 Runtime installation is unnecessary.

1 Go to the Agilent web site at: "www.agilent.com/find/toolkit"

2 On the T&M Toolkit 2.1 with Test Automation page's **Technical Support** tab, choose **Drivers & Software**.

3 On the Drivers & Software tab page, download:

- **Agilent T&M Toolkit 2.1 with Test Automation** – Download the executable and readme file.

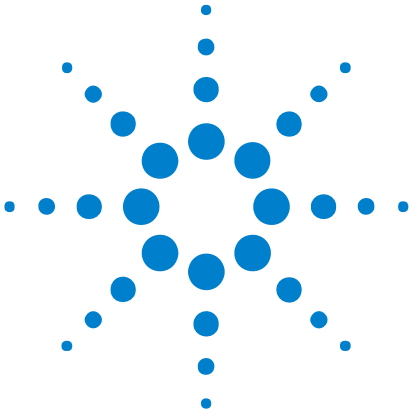
  This is a large file, unfortunately, but it contains the Agilent T&M Toolkit 2.1 Runtime installation.

- **Agilent T&M Toolkit 2.1 with Test Automation Redistributable Service Pack 1** – Download the executable and readme file.

4 Run the Agilent T&M Toolkit 2.1 with Test Automation executable file. When the setup window appears, scroll down, and click the **Install Agilent T&M Toolkit Runtime** button.

5 Run the Agilent T&M Toolkit 2.1 with Test Automation Redistributable Service Pack 1 executable file.

After installing the Agilent T&M Toolkit 2.1 Runtime (or if Agilent VEE is already installed), you can use the Agilent Instrument Explorer to edit the IVI Configuration Store. See “Editing the IVI Configuration Store with Agilent Instrument Explorer" on page 18.



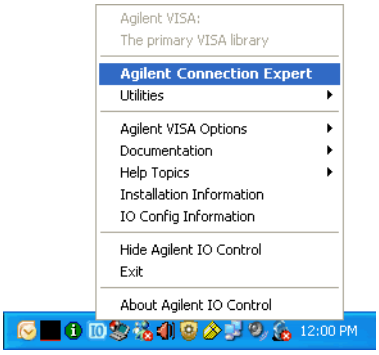
2 Performing Additional Set Up

Adding Your Oscilloscope Using Agilent Connection Expert

This procedure only needs to be performed once for each oscilloscope that will be controlled.

NOTE The menus and dialogs shown here may differ slightly depending on the version of the Agilent IO Libraries Suite.

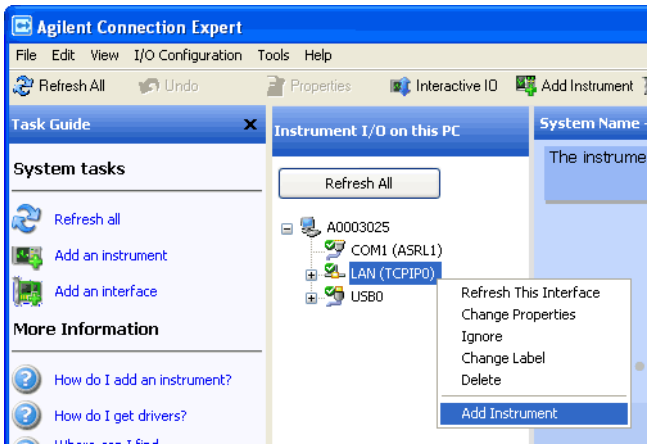
1 On the controller PC, choose **Start>All Programs>Agilent IO Libraries Suite>Agilent Connection Expert** from the Windows Start menu. Or, click on the Agilent IO Control icon in the taskbar, and choose Agilent Connection Expert from the popup menu.



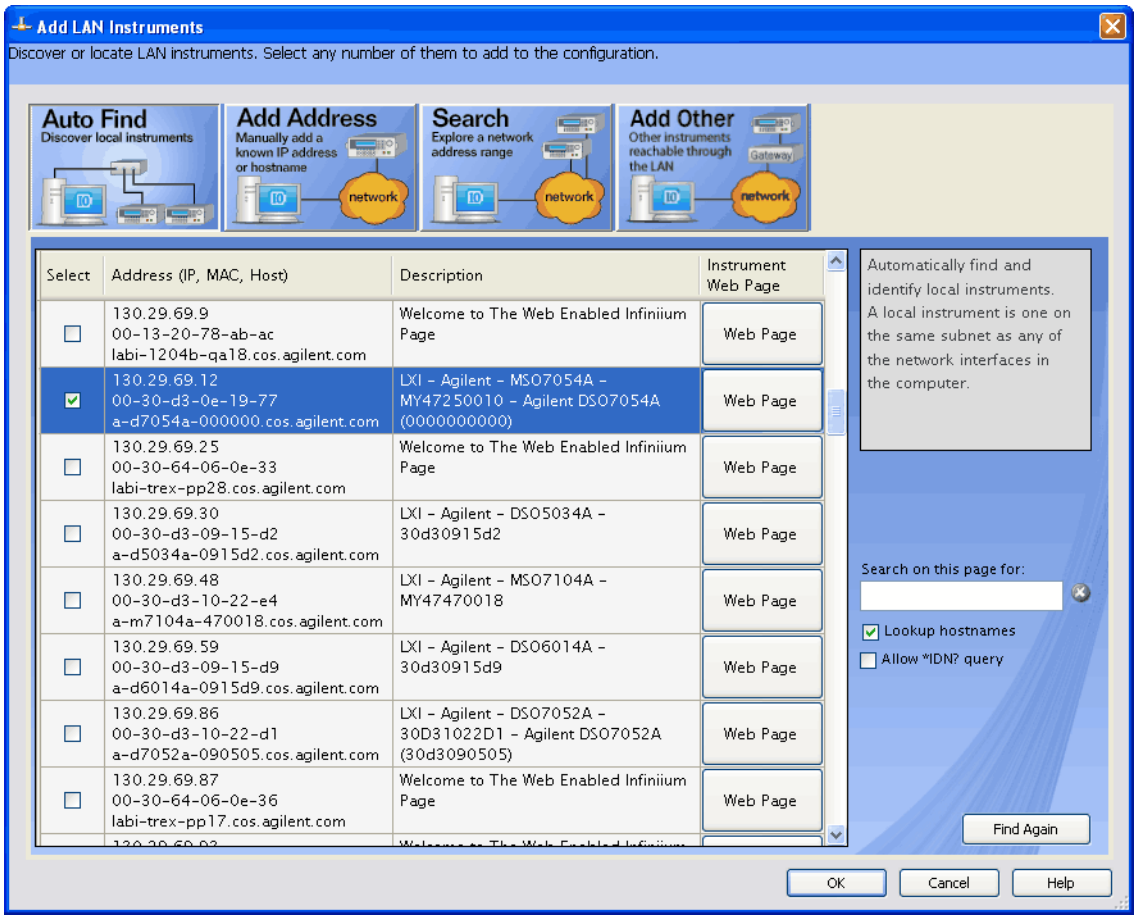
2 In the Agilent Connection Expert application, instruments connected to the controller's USB and GPIB interfaces should automatically appear. (You can click Refresh All to update the list of instruments on these interfaces.)

You must manually add instruments on LAN interfaces:

a Right-click on the LAN interface, choose **Add Instrument** from the popup menu, and click **OK** in the resulting dialog (because the desired interface is already selected).



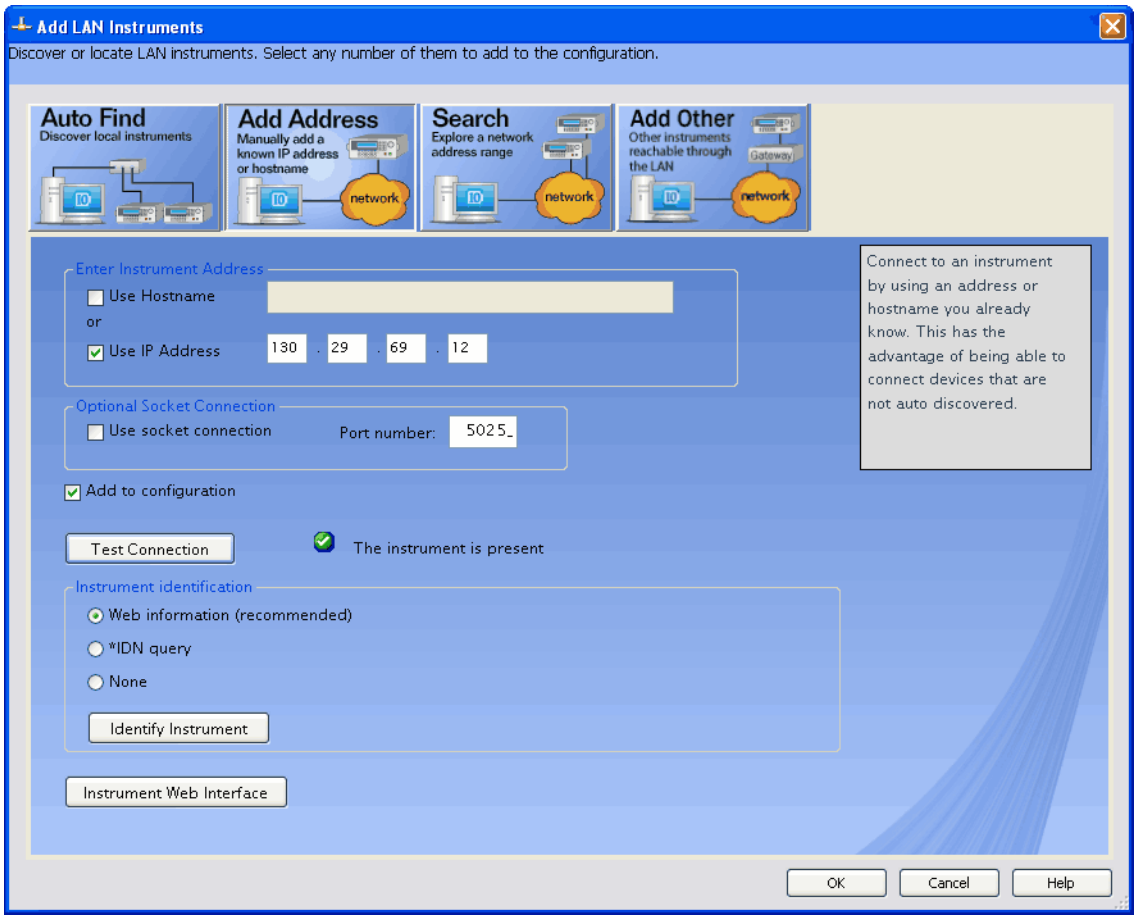
b If the oscilloscope is on the same subnet, select it and click **OK**.



Otherwise, click **Add Address** (or if you have a version of the IO Libraries that doesn't have Auto Find, select the LAN interface and click **OK**).

c In the next dialog, select either **Hostname** or **IP address**, and enter the oscilloscope's hostname or IP address.

d Click **Test Connection**.



e If the instrument is successfully opened, click **OK** to close the dialog. If the instrument is not opened successfully, go back and verify the LAN connections and the oscilloscope setup.

3 In the Agilent Connection Expert application, choose **File>Exit** from the menu to exit the application.

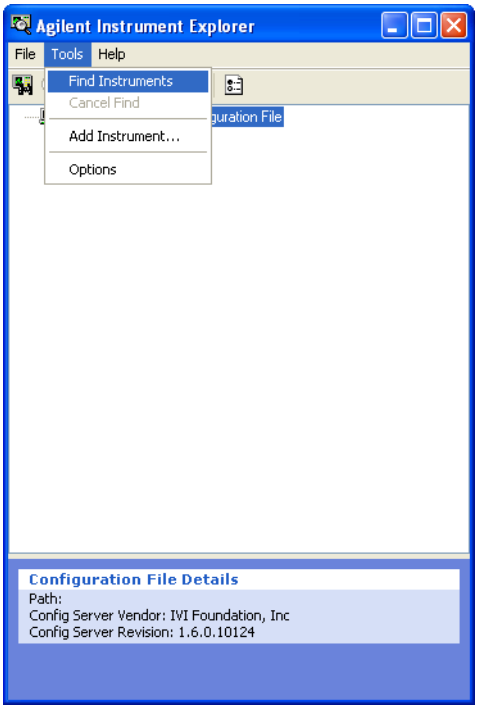
Editing the IVI Configuration Store with Agilent Instrument Explorer

This procedure only needs to be performed once for each oscilloscope that will be controlled.

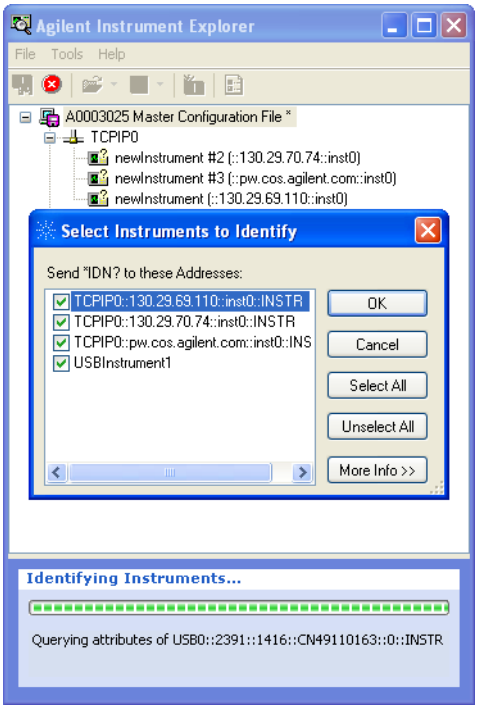
Before performing this procedure, make sure you have added your oscilloscope using the Agilent Connection Expert (see "Adding Your Oscilloscope Using Agilent Connection Expert" on page 14).

1 On the controller PC, choose **Start>All Programs>Agilent T&M Toolkit 2.1>Agilent Instrument Explorer** from the Windows Start menu.

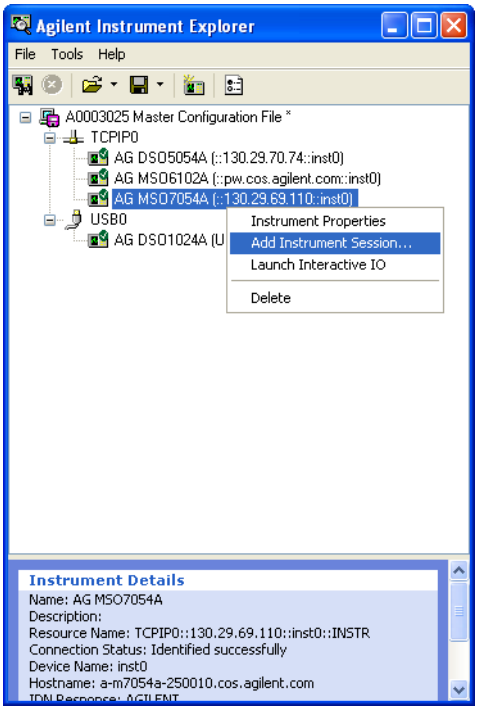
2 In the Agilent Instrument Explorer window, choose **Tools>Find Instruments**.



3 In the Select Instruments to Identify dialog, click **OK**.



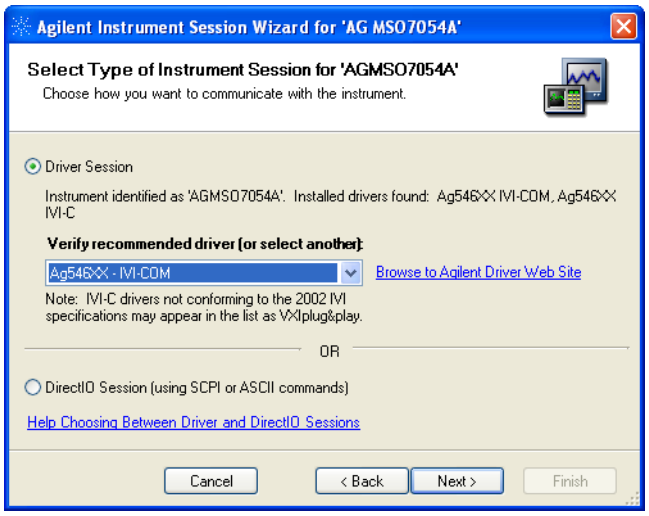
4 Right-click on the your oscilloscope, and choose **Add Instrument Session...** from the popup menu.



5 In the Agilent Instrument Session Wizard:

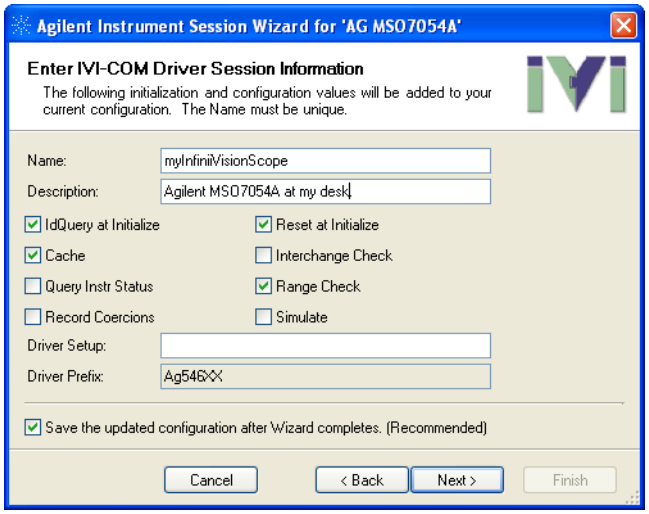
a Click **Next >**.

b Select **Driver Session** and select the **Ag546XX - IVI-COM** driver.



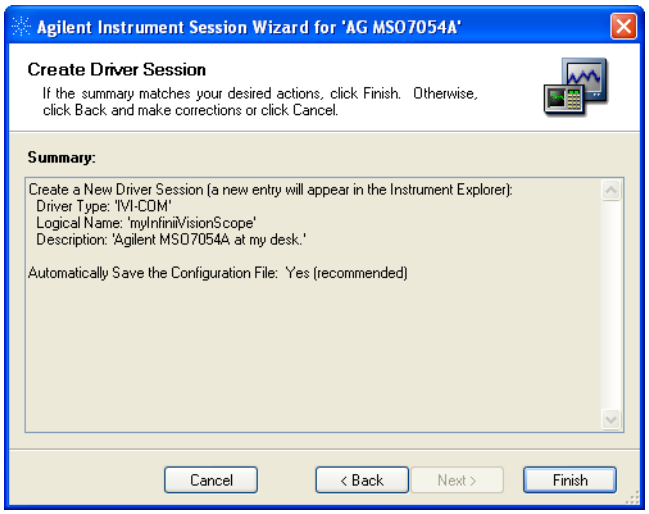
Then, click **Next >**.

c Enter the session **Name** and **Description**.

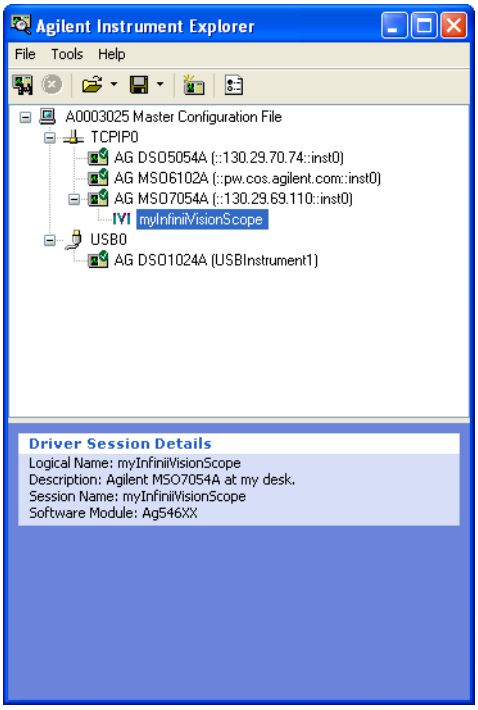


Then, click **Next >**.

d Click **Finish**.



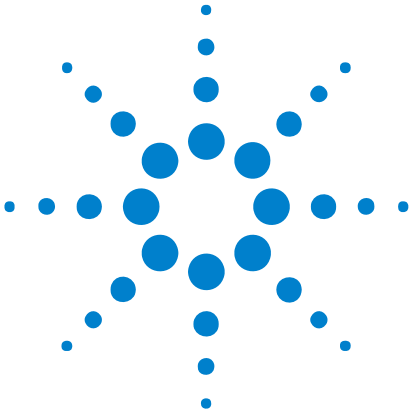
6 Select the added session and view the details.



Note the **Logical Name**. This is the name, saved in the IVI Configuration Store, that can be used when creating an instance of the IVI-COM driver in your programs.

The IVI Configuration Store is typically located at: C:\Program Files\IVI\Data\IviConfigurationStore.xml

7 Choose **File>Exit** to close the Agilent Instrument Explorer window



3 Quick Start Examples

Visual C++ Quick Start

To compile and run this example in Microsoft Visual Studio 2008:

1. Open Visual Studio.
2. Create a new Visual C++, Win32, Win32 Console Application project.
3. In the Win32 Application Wizard, click **Finish** (precompiled header option is selected by default).
4. Cut-and-paste the code that follows into a file named "QuickStart.cpp" in the project directory.
5. In Visual Studio 2008, right-click the Source Files folder, choose **Add > Add Existing Item...**, select the QuickStart.cpp file, and click **Add**.
6. Choose **Project > Properties...**. In the Property Pages dialog, update these project settings:
   a. Under Configuration Properties, Linker, Input, add "visa32.lib" to the Additional Dependencies field.
   b. Under Configuration Properties, C/C++, Code Generation, select Multi-threaded DLL for the Runtime Library field.
   c. Click **OK** to close the Property Pages dialog.
7. Add the executable files search paths:
   a. Choose **Tools > Options...**.
   b. In the Options dialog, select **VC++ Directories** under Projects and Solutions.
   c. Show directories for **Executable files**, and add these directories:
      - C:\Program Files\IVI\Bin
      - $(VXIPNPPATH)VisaCom
   d. Click **OK** to close the Options dialog.
8. Build and run the program.

```
/*
 * IVI-COM Driver for InfiniiVision Oscilloscopes, C++
 * ===================================================================
 */

#include "stdafx.h"

#import "IviDriverTypeLib.dll" no_namespace
#import "IviScopeTypeLib.dll" no_namespace
#import "IviSessionFactory.dll" no_namespace

// Function prototypes.
void DisplayDriverIdentity(IIviScopePtr spScope);
void DisplayInstrumentIdentity(IIviScopePtr spScope);
void InitializeScope(IIviScopePtr spScope);
```

```
void CaptureData(IIviScopePtr spScope);
void AnalyzeData(IIviScopePtr spScope);
void ReadInstrumentErrors(IIviDriverPtr spDrvr);

/* Main Program
 * ------------------------------------------------------------------- */
int _tmain(int argc, _TCHAR* argv[])
{
  HRESULT hr = ::CoInitialize(NULL);

  try
  {
    _bstr_t logicalName = "myInfiniiVisionScope";

    // Create the driver.
    IIviSessionFactoryPtr spFactory(__uuidof(IviSessionFactory));
    IIviScopePtr spScope = spFactory->CreateDriver(logicalName);

    // Display driver identity properties (no initialization req'd).
    DisplayDriverIdentity(spScope);

    // If VARIANT_TRUE, this will query the instrument model and fail
    // initialization if the model is not supported by the driver.
    VARIANT_BOOL idQuery = VARIANT_FALSE;

    // If VARIANT_TRUE, the instrument is reset at initialization.
    VARIANT_BOOL reset = VARIANT_FALSE;

    // Set up IVI-defined initialization options.
    _bstr_t standardInitOptions = "Cache=true, \
        InterchangeCheck=false, QueryInstrStatus=true, \
        RangeCheck=true, RecordCoercions=false, Simulate=false";

    // Set up driver-specific initialization options.
    _bstr_t driverSetupOptions =
        "DriverSetup= Model=Agilent546XX, Trace=false";

    // Initialize the driver.
    spScope->Initialize(logicalName, idQuery, reset,
        standardInitOptions + _bstr_t(_T(",")) + driverSetupOptions);

    // Display instrument identity properties (initialization req'd).
    DisplayInstrumentIdentity(spScope);

    try
    {
      // Initialize the oscilloscope to a known state.
      InitializeScope(spScope);

      // Capture data.
      CaptureData(spScope);

      // Analyze the captured data.
      AnalyzeData(spScope);
    }
    catch (_com_error& ex)
    {
```

```
      if (ex.Error() == E_IVI_INSTRUMENT_STATUS)
      {
        // ErrorQuery should give us more information.
        ReadInstrumentErrors(spScope);
      }
      else
      {
        // Print the exception.
        wprintf(L"Exception: %s\n", ex.Description());
      }
    }

    // Close the driver if initialized.
    if(spScope->GetInitialized() == VARIANT_TRUE)
    {
      printf("\nClosing initialized driver...\n");
      spScope->Close();
    }
    if(spScope) spScope.Release();
  }
  catch (_com_error& e)
  {
    // There was some problem creating the driver or initializing
    // it.  There is no need to call Close() and we cannot call
    // ErrorQuery() because the driver is not initialized.
    wprintf(L"Exception: %s\n", e.Description());
  }

  // Note that by this point all COM objects have been released
  // because all the smart pointers go out of scope at the end of
  // the outermost try block.
  // This is important, no COM calls can be made after CoUninitialize.
  ::CoUninitialize();   // Matching call to CoInitialize.
  printf("\nDone - Press Enter to Exit");
  getchar();
  return 0;
}

/* ================================================================
 * Display driver identity properties (no initialization req'd).
 * ----------------------------------------------------------------
 */
void DisplayDriverIdentity(IIviScopePtr spScope)
{
  BSTR identifier = spScope->Identity->Identifier;
  wprintf(L"Driver identifier: %s\n", identifier);

  BSTR description = spScope->Identity->Description;
  wprintf(L"Driver description: %s\n", description);

  BSTR revision = spScope->Identity->Revision;
  wprintf(L"Driver revision: %s\n", revision);

  BSTR vendor = spScope->Identity->Vendor;
  wprintf(L"Driver vendor: %s\n", vendor);
}
```

```
/* ====================================================================
 * Display instrument identity properties (initialization req'd).
 * --------------------------------------------------------------------
 */
void DisplayInstrumentIdentity(IIviScopePtr spScope)
{
  BSTR instModel = spScope->Identity->InstrumentModel;
  wprintf(L"Instrument model: %s\n", instModel);

  BSTR instFwRev = spScope->Identity->InstrumentFirmwareRevision;
  wprintf(L"Instrument firmware revision: %s\n", instFwRev);

  BSTR instMfr = spScope->Identity->InstrumentManufacturer;
  wprintf(L"Instrument manufacturer: %s\n", instMfr);
}

/* ====================================================================
 * Initialize the oscilloscope to a known state.
 * --------------------------------------------------------------------
 */
void InitializeScope(IIviScopePtr spScope)
{
  // Place the instrument into a known state (*RST, etc.).
  spScope->Utility->Reset();
}

/* ====================================================================
 * Capture data.
 * --------------------------------------------------------------------
 */
void CaptureData(IIviScopePtr spScope)
{
  // Configure the trigger type and holdoff.  Type can be edge,
  // pulse width, glitch, TV, or ACLine (runt and immediate
  // not supported in InfiniiVision oscilloscopes).
  // ------------------------------------------------------------------

  // Set up edge trigger.
  double dHoldoff = 0.000000060;   // 60ns to 10s.
  spScope->Trigger->Configure(IviScopeTriggerEdge, dHoldoff);

  spScope->Trigger->Edge->Configure(L"Channel1", 2.5,
    IviScopeTriggerSlopePositive);
  wprintf(L"Trigger edge source: %s\n",
    (BSTR)spScope->Trigger->Source);
  wprintf(L"Trigger edge level: %.15g\n", spScope->Trigger->Level);
  wprintf(L"Trigger edge slope: %d\n", spScope->Trigger->Edge->Slope);

  // Trigger coupling.
  spScope->Trigger->Coupling = IviScopeTriggerCouplingDC;
  wprintf(L"Trigger coupling: %d\n", spScope->Trigger->Coupling);

  // Trigger modifier (Auto vs. Normal).
  spScope->Trigger->Modifier = IviScopeTriggerModifierNone;
  wprintf(L"Trigger modifier: %d\n", spScope->Trigger->Modifier);

  // Set vertical range and offset.
```

```
// ------------------------------------------------------------------
IIviScopeChannelPtr spScopeCh;
spScopeCh = spScope->Channels->Item[_T("Channel1")];

// Channel coupling.
spScopeCh->Coupling = IviScopeVerticalCouplingDC;
wprintf(L"Ch1 coupling: %d\n", spScopeCh->Coupling);

// Make sure channel is enabled.
spScopeCh->Enabled = VARIANT_TRUE;
wprintf(L"Ch1 enabled: %s\n",
  ((spScopeCh->Enabled == VARIANT_TRUE) ? L"True" : L"False"));

// Vertical range.
spScopeCh->Range = 8.0;
wprintf(L"Ch1 vertical range: %.2f\n", spScopeCh->Range);

// Vertical offset.
spScopeCh->Offset = 2.5;
wprintf(L"Ch1 vertical offset: %.2f\n", spScopeCh->Offset);

// Set horizontal scale and offset.
// ------------------------------------------------------------------
double dRange = 0.0000001;   // Horiz range.
int nSamples = 1000;   // Number of points in waveform.
double dOffset = -0.00000005;   // Horiz position.

spScope->Acquisition->ConfigureRecord(dRange, nSamples, dOffset);
wprintf(L"Record length (points): %d\n",
  spScope->Acquisition->RecordLength);
wprintf(L"Record start time: %E\n",
  spScope->Acquisition->StartTime);
wprintf(L"Time of record: %E\n",
  spScope->Acquisition->TimePerRecord);

// Set the acquisition type. Type can be Normal, PeakDetect,
// HiRes, or Average (Envelope not supported in InfiniiVision
// oscilloscopes).
// ------------------------------------------------------------------

// Interpolation setting.
spScope->Acquisition->Interpolation = IviScopeInterpolationSineX;
wprintf(L"Interpolation: %d\n",
  spScope->Acquisition->Interpolation);

// Averaging.
spScope->Acquisition->Type = IviScopeAcquisitionTypeAverage;
wprintf(L"Acquisition type: %d\n",
  spScope->Acquisition->Type);

spScope->Acquisition->NumberOfAverages = 1024;
wprintf(L"Number of averages: %d\n",
  spScope->Acquisition->NumberOfAverages);

// Or, configure by performing Auto-Scale.
// ------------------------------------------------------------------
//spScope.Measurements.AutoSetup();
```

```
  // Acquire data.
  // -------------------------------------------------------------------
  spScope->Measurements->Initiate();
  wprintf(L"Measurements status: %d\n",
    spScope->Measurements->Status());
  wprintf(L"Sample mode: %d\n", spScope->Acquisition->SampleMode);
  wprintf(L"Points per second: %E\n",
    spScope->Acquisition->SampleRate);
}

/* ====================================================================
 * Analyze the captured data.
 * -------------------------------------------------------------------
 */
void AnalyzeData(IIviScopePtr spScope)
{

  // Measurements and waveforms are available for each channel.
  for (int i = 1; i <= spScope->Measurements->Count; i++)
  {
    // measName is the name of a channel.
    BSTR measName = spScope->Measurements->Name[i];

    // Operate on Measurement item.
    IIviScopeChannelPtr spScopeCh;
    spScopeCh = spScope->Channels->Item[measName];
    IIviScopeMeasurementPtr spScopeMeas;
    spScopeMeas = spScope->Measurements->Item[measName];

    if (spScopeCh->Enabled)
    {
      // Get measurement values.
      // -----------------------------------------------------------
      double dValue = 0.0;

      spScopeMeas->FetchWaveformMeasurement(
        IviScopeMeasurementRiseTime, &dValue);
      wprintf(L"%s rise time: %E\n", measName, dValue);

      spScopeMeas->FetchWaveformMeasurement(
        IviScopeMeasurementVoltagePeakToPeak, &dValue);
      wprintf(L"%s voltage pk-pk: %E\n", measName, dValue);

      // Get waveform data.
      // -----------------------------------------------------------
      SAFEARRAY* psaWaveform = 0;
      double dInitialX   = 0.0;
      double dXIncrement = 0.0;

      spScopeMeas->FetchWaveform(&psaWaveform, &dInitialX,
        &dXIncrement);

      long lBound, uBound;
      double * pData;

      SafeArrayGetLBound(psaWaveform, 1, &lBound);
```

```
      SafeArrayGetUBound(psaWaveform, 1, &uBound);
      SafeArrayAccessData(psaWaveform, (void**)&pData);
      wprintf(L"Data points fetched: %i\n", (uBound-lBound+1));

      // Open file for output.
      FILE *fp;

      _bstr_t strPath = "c:\\scope\\data\\waveform_data.csv";
      fp = fopen(strPath, "wb");

      // Output waveform data in CSV format.
      for (int i = 0; i < uBound-lBound+1; i++)
      {
        // Write time value, voltage value.
        fprintf(fp, "%E, %6f\n",
          dInitialX + ((float)i * dXIncrement), pData[i]);
      }
      SafeArrayUnaccessData(psaWaveform);
      SafeArrayDestroy(psaWaveform);

      // Close output file.
      fclose(fp);
      wprintf(L"Waveform data written to %s\n", strPath.GetBSTR());
    }
  }
}

/* ========================================================================
* Read instrument errors.
* ------------------------------------------------------------------------
*/
void ReadInstrumentErrors(IIviDriverPtr spDrvr)
{
  // Read instrument error queue until empty.
  long errNum = 999;
  BSTR errMsg = NULL;

  printf("\n");
  while (errNum != 0)
  {
    spDrvr->Utility->ErrorQuery(&errNum, &errMsg);
    wprintf(L"ErrorQuery: %d, %s\n", errNum, errMsg);
  }
  // ErrorQuery() allocates memory for errMsg so we must free it.
  if (errMsg != NULL) ::SysFreeString(errMsg);
}
```

Visual C# Quick Start

To compile and run this example in Microsoft Visual Studio 2008:

1. Open Visual Studio.
2. Create a new Visual C#, Windows, Console Application project.
3. Cut-and-paste the code that follows into the C# source file.
4. Add references to the IVI-COM driver DLLs:
   a. Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment.
   b. Choose **Add Reference...**.
   c. In the Add Reference dialog, select the **COM** tab.
   d. Select these libraries:
      - IviDriver 1.0 Type Library (for inherent capabilities)
      - IviScope 3.0 Type Library (for class-compliant interface)
      - IviSessionFactory 1.0 Type Library (facilitates interchangeable COM applications)
      - IVI Agilent546XX 1.3 Type Library (for instrument-specific interface)

      Then, click **OK**.
5. Build and run the program.

```
/*
 * IVI-COM Driver for InfiniiVision Oscilloscopes, C#
 * ================================================================
 */

using System;
using System.IO;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.InteropServices;
using Ivi.Driver.Interop;
using Ivi.Scope.Interop;   // Class-compliant.
// using Agilent.Agilent546XX.Interop;   // Instrument-specific.
using Ivi.SessionFactory.Interop;

namespace ClientApp
{
  public class App
  {
    [STAThread]
    public static void Main(string[] args)
    {
```

```
try
{
  string logicalName = "myInfiniiVisionScope";

  IIviSessionFactory factory = new IviSessionFactoryClass();
  IIviDriver driver =
      (IIviDriver)factory.CreateDriver(logicalName);

  // Display driver identity - Initialize not required.
  DisplayDriverIdentity(driver);

  // Set up IVI-defined initialization options.
  string standardInitOptions =
      "Cache=true, " +
      "InterchangeCheck=false, " +
      "QueryInstrStatus=true, " +
      "RangeCheck=true, " +
      "RecordCoercions=false, " +
      "Simulate=false";

  // Setup driver-specific initialization options.
  string driverSetupOptions =
    "DriverSetup= Model=Agilent546XX, Trace=false";

  driver.Initialize(logicalName, false, false,
      standardInitOptions + ", " + driverSetupOptions);

  try
  {
    // Display instrument identity - Initialize required.
    DisplayInstrumentIdentity(driver);

    IIviScope myScope = (IIviScope)driver;

    // Initialize the oscilloscope to a known state.
    InitializeScope(myScope);

    // Capture data.
    CaptureData(myScope);

    // Analyze the captured data.
    AnalyzeData(myScope);
  }
  catch (COMException ex)
  {
    if (ex.ErrorCode ==
        (int)IviDriver_ErrorCodes.E_IVI_INSTRUMENT_STATUS)
    {
      // ErrorQuery should give more information.
      ReadInstrumentErrors(driver);
    }
    else
    {
      // Print the exception.
      Console.WriteLine("Exception: {0}", ex.Message);
    }
  }
```

```
    // Close driver.
    driver.Close();
  }
  catch (Exception e)
  {
    Console.WriteLine(e.Message);
  }

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

/* ============================================================
 * Display driver identity properties (no initialization req'd).
 * ------------------------------------------------------------
 */
private static void DisplayDriverIdentity(IIviDriver driver)
{
  string identifier = driver.Identity.Identifier;
  Console.WriteLine("Driver identifier: {0}", identifier);

  string description = driver.Identity.Description;
  Console.WriteLine("Driver description: {0}", description);

  string revision = driver.Identity.Revision;
  Console.WriteLine("Driver revision: {0}", revision);

  string vendor = driver.Identity.Vendor;
  Console.WriteLine("Driver vendor: {0}", vendor);

}

/* ============================================================
 * Display instrument identity properties (initialization req'd).
 * ------------------------------------------------------------
 */
private static void DisplayInstrumentIdentity(IIviDriver driver)
{
  string instModel = driver.Identity.InstrumentModel;
  Console.WriteLine("Instrument model: {0}", instModel);

  string instFirmwareRevision =
      driver.Identity.InstrumentFirmwareRevision;
  Console.WriteLine("Instrument firmware revision: {0}",
      instFirmwareRevision);

  string instManufacturer =
      driver.Identity.InstrumentManufacturer;
  Console.WriteLine("Instrument manufacturer: {0}",
      instManufacturer);

}

/* ============================================================
 * Initialize the oscilloscope to a known state.
 * ------------------------------------------------------------
```

```
 */
private static void InitializeScope(IIviScope myScope)
{
  // Place the instrument into a known state (*RST, etc.).
  myScope.Utility.Reset();
}

/* ================================================================
 * Capture data.
 * ----------------------------------------------------------------
 */
private static void CaptureData(IIviScope myScope)
{
  // Configure the trigger type and holdoff.  Type can be edge,
  // pulse width, glitch, TV, or ACLine (runt and immediate
  // not supported in InfiniiVision oscilloscopes).
  // ----------------------------------------------------------------

  // Set up edge trigger.
  double dHoldoff = 0.000000060;   // 60ns to 10s.
  myScope.Trigger.Configure(
    IviScopeTriggerTypeEnum.IviScopeTriggerEdge, dHoldoff);

  myScope.Trigger.Edge.Configure("Channel1", 2.5,
      IviScopeTriggerSlopeEnum.IviScopeTriggerSlopePositive);
  Console.WriteLine("Trigger edge source: " +
      myScope.Trigger.Source.ToString());
  Console.WriteLine("Trigger edge level: " +
      myScope.Trigger.Level.ToString());
  Console.WriteLine("Trigger edge slope: " +
      myScope.Trigger.Edge.Slope.ToString());

  // Trigger coupling.
  myScope.Trigger.Coupling =
      IviScopeTriggerCouplingEnum.IviScopeTriggerCouplingDC;
  Console.WriteLine("Trigger coupling: " +
      myScope.Trigger.Coupling.ToString());

  // Trigger modifier (Auto vs. Normal).
  myScope.Trigger.Modifier =
      IviScopeTriggerModifierEnum.IviScopeTriggerModifierNone;
  Console.WriteLine("Trigger modifier: " +
      myScope.Trigger.Modifier.ToString());

  // Set vertical range and offset.
  // ----------------------------------------------------------------
  IIviScopeChannel myScopeCh;
  myScopeCh = myScope.Channels.get_Item("Channel1");

  // Channel coupling.
  myScopeCh.Coupling =
      IviScopeVerticalCouplingEnum.IviScopeVerticalCouplingDC;
  Console.WriteLine("Ch1 coupling: " +
      myScopeCh.Coupling.ToString());

  // Make sure channel is enabled.
  myScopeCh.Enabled = true;
```

```
Console.WriteLine("Ch1 enabled: " +
    myScopeCh.Enabled.ToString());

// Vertical range.
myScopeCh.Range = 8.0;
Console.WriteLine("Ch1 vertical range: " +
    myScopeCh.Range.ToString());

// Vertical offset.
myScopeCh.Offset = 2.5;
Console.WriteLine("Ch1 vertical offset: " +
    myScopeCh.Offset.ToString());

// Set horizontal scale and offset.
// -------------------------------------------------------------
double dRange = 0.0000001;   // Horiz range.
int nSamples = 1000;   // Number of points in waveform.
double dOffset = -0.00000005;   // Horiz position.

myScope.Acquisition.ConfigureRecord(dRange, nSamples, dOffset);
Console.WriteLine("Record length (points): {0:D}",
    myScope.Acquisition.RecordLength);
Console.WriteLine("Record start time: {0:E}",
    myScope.Acquisition.StartTime);
Console.WriteLine("Time of record: {0:E}",
    myScope.Acquisition.TimePerRecord);

// Set the acquisition type. Type can be Normal, PeakDetect,
// HiRes, or Average (Envelope not supported in InfiniiVision
// oscilloscopes).
// -------------------------------------------------------------

// Interpolation setting.
myScope.Acquisition.Interpolation =
    IviScopeInterpolationEnum.IviScopeInterpolationSineX;
Console.WriteLine("Interpolation: " +
    myScope.Acquisition.Interpolation.ToString());

// Averaging.
myScope.Acquisition.Type =
    IviScopeAcquisitionTypeEnum.IviScopeAcquisitionTypeAverage;
Console.WriteLine("Acquisition type: " +
    myScope.Acquisition.Type.ToString());

myScope.Acquisition.NumberOfAverages = 1024;
Console.WriteLine("Number of averages: {0}",
    myScope.Acquisition.NumberOfAverages);

// Or, configure by performing Auto-Scale.
// -------------------------------------------------------------
//myScope.Measurements.AutoSetup();

// Acquire data.
// -------------------------------------------------------------
myScope.Measurements.Initiate();
Console.WriteLine("Measurements status: " +
    myScope.Measurements.Status());
```

```
  Console.WriteLine("Sample mode: " +
      myScope.Acquisition.SampleMode.ToString());
  Console.WriteLine("Points per second: {0:E}",
      myScope.Acquisition.SampleRate);
}

/* ================================================================
 * Analyze the captured data.
 * ----------------------------------------------------------------
 */
private static void AnalyzeData(IIviScope myScope)
{
  // Set up measurements hashtable.
  Hashtable htMeasurements = new Hashtable();
  htMeasurements.Add(
    IviScopeMeasurementEnum.IviScopeMeasurementRiseTime,
    "rise time");
  htMeasurements.Add(
    IviScopeMeasurementEnum.IviScopeMeasurementVoltagePeakToPeak,
    "voltage pk-pk");

  // Measurements and waveforms are available for each channel.
  for (int i = 1; i <= myScope.Measurements.Count; i++)
  {
    // measName is the name of a channel.
    string measName = myScope.Measurements.get_Name(i).ToString();

    // Operate on Measurement item.
    IIviScopeChannel myScopeCh;
    myScopeCh = myScope.Channels.get_Item(measName);
    IIviScopeMeasurement myScopeMeas;
    myScopeMeas = myScope.Measurements.get_Item(measName);

    if (myScopeCh.Enabled)
    {
      // Get measurement values.
      // ------------------------------------------------------
      ICollection cMeasurements = htMeasurements.Keys;
      foreach (IviScopeMeasurementEnum eMeas in cMeasurements)
      {
        // Display measurement.
        double dValue = 0.0;
        myScopeMeas.FetchWaveformMeasurement(eMeas, ref(dValue));
        Console.WriteLine("{0} {1}: {2:E}",
            measName, htMeasurements[eMeas], dValue);
      }

      // Get waveform data.
      // ------------------------------------------------------
      double[] waveformData = new double[1000];
      double dInitialX = 0.0;
      double dXIncrement = 0.0;

      myScopeMeas.FetchWaveform(ref(waveformData),
          ref(dInitialX), ref(dXIncrement));
      Console.WriteLine("Data points fetched: {0}",
          waveformData.Length);
```

```
            // Set up output file:
            string strPath;
            strPath = "c:\\scope\\data\\waveform_data.csv";
            if (File.Exists(strPath)) File.Delete(strPath);

            // Open file for output.
            StreamWriter writer = File.CreateText(strPath);

            // Output waveform data in CSV format.
            for (int j = 0; j < waveformData.Length; j++)
              writer.WriteLine("{0:E}, {1:F6}",
                  (dInitialX + ((float)j * dXIncrement)),
                  waveformData[j]);

            // Close output file.
            writer.Close();
            Console.WriteLine("Waveform data written to {0}", strPath);
          }
        }
      }

      /* ================================================================
       * Read instrument errors.
       * ----------------------------------------------------------------
       */
      static void ReadInstrumentErrors(IIviDriver driver)
      {
        // Read instrument error queue until its empty.
        int errCode = 0;
        string errMsg = null;

        //Console.WriteLine();
        do
        {
          driver.Utility.ErrorQuery(ref errCode, ref errMsg);
          if (errCode != 0)
          {
            Console.WriteLine("ErrorQuery: {0}, {1}", errCode, errMsg);
          }
        }
        while (errCode != 0);
      }
    }
  }
}
```

Visual Basic .NET Quick Start

To compile and run this example in Microsoft Visual Studio 2008:

1. Open Visual Studio.
2. Create a new Visual Basic, Windows, Console Application project.
3. Cut-and-paste the code that follows into the C# source file.
4. Add a reference to the VISA COM 3.0 Type Library:
   a. Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment.
   b. Choose **Add Reference...**.
   c. In the Add Reference dialog, select the **COM** tab.
   d. Select **VISA COM 3.0 Type Library**; then click **OK**.
   e. Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment and choose **Properties**; then, select "InfiniiVision.VisaComInstrumentApp" as the **Startup object**.
5. Add references to the IVI-COM driver DLLs:
   a. Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment.
   b. Choose **Add Reference...**.
   c. In the Add Reference dialog, select the **COM** tab.
   d. Select these libraries:
      - IviDriver 1.0 Type Library (for inherent capabilities)
      - IviScope 3.0 Type Library (for class-compliant interface)
      - IviSessionFactory 1.0 Type Library (facilitates interchangeable COM applications)
      - IVI Agilent546XX 1.3 Type Library (for instrument-specific interface)

      Then, click **OK**.
6. Build and run the program.

```
'
' IVI-COM Driver for InfiniiVision Oscilloscopes, Visual Basic .NET
' ====================================================================

Imports System
Imports System.IO
Imports System.Collections
Imports System.Collections.Generic
```

```
Imports System.Linq
Imports System.Text
Imports System.Runtime.InteropServices
Imports Ivi.Driver.Interop
Imports Ivi.Scope.Interop   ' Class-compliant.
' Imports Agilent.Agilent546XX.Interop   ' Instrument-specific.
Imports Ivi.SessionFactory.Interop

Namespace ClientApp
  Public Class App
    <STAThread()> _
    Public Shared Sub Main(ByVal args As String())
      Try
        Dim logicalName As String = "myInfiniiVisionScope"

        Dim factory As IIviSessionFactory = _
            New IviSessionFactoryClass()
        Dim driver As IIviDriver = _
            DirectCast(factory.CreateDriver(logicalName), IIviDriver)

        ' Display driver identity - Initialize not required.
        DisplayDriverIdentity(driver)

        ' Set up IVI-defined initialization options.
        Dim standardInitOptions As String = _
            "Cache=true, " & _
            "InterchangeCheck=false, " & _
            "QueryInstrStatus=true, " & _
            "RangeCheck=true, " & _
            "RecordCoercions=false, " & _
            "Simulate=false"

        ' Setup driver-specific initialization options.
        Dim driverSetupOptions As String = _
            "DriverSetup= Model=Agilent546XX, Trace=false"

        driver.Initialize(logicalName, False, False, _
            (standardInitOptions & ", ") + driverSetupOptions)

        Try
          ' Display instrument identity - Initialize required.
          DisplayInstrumentIdentity(driver)

          Dim myScope As IIviScope = DirectCast(driver, IIviScope)

          ' Initialize the oscilloscope to a known state.
          InitializeScope(myScope)

          ' Capture data.
          CaptureData(myScope)

          ' Analyze the captured data.
          AnalyzeData(myScope)

        Catch ex As COMException
          If ex.ErrorCode = _
              CInt(IviDriver_ErrorCodes.E_IVI_INSTRUMENT_STATUS) Then
```

```
        ' ErrorQuery should give more information.
        ReadInstrumentErrors(driver)
      Else
        ' Print the exception.
        Console.WriteLine("Exception: {0}", ex.Message)
      End If
    End Try

    ' Close driver.
    driver.Close()

  Catch e As Exception
    Console.WriteLine(e.Message)
  End Try

  Console.WriteLine("Done - Press Enter to Exit")
  Console.ReadLine()
End Sub

' ================================================================
' Display driver identity properties (no initialization req'd).
' ----------------------------------------------------------------
Private Shared Sub _
    DisplayDriverIdentity(ByVal driver As IIviDriver)
  Dim identifier As String = driver.Identity.Identifier
  Console.WriteLine("Driver identifier: {0}", identifier)

  Dim description As String = driver.Identity.Description
  Console.WriteLine("Driver description: {0}", description)

  Dim revision As String = driver.Identity.Revision
  Console.WriteLine("Driver revision: {0}", revision)

  Dim vendor As String = driver.Identity.Vendor

  Console.WriteLine("Driver vendor: {0}", vendor)
End Sub

' ================================================================
' Display instrument identity properties (initialization req'd).
' ----------------------------------------------------------------
Private Shared Sub _
    DisplayInstrumentIdentity(ByVal driver As IIviDriver)
  Dim instModel As String = driver.Identity.InstrumentModel
  Console.WriteLine("Instrument model: {0}", instModel)

  Dim instFirmwareRevision As String = _
      driver.Identity.InstrumentFirmwareRevision
  Console.WriteLine("Instrument firmware revision: {0}", _
      instFirmwareRevision)

  Dim instManufacturer As String = _
      driver.Identity.InstrumentManufacturer

  Console.WriteLine("Instrument manufacturer: {0}", _
      instManufacturer)
End Sub
```

```
' ====================================================================
' Initialize the oscilloscope to a known state.
' --------------------------------------------------------------------
Private Shared Sub InitializeScope(ByVal myScope As IIviScope)
  ' Place the instrument into a known state (*RST, etc.).
  myScope.Utility.Reset()
End Sub

' ====================================================================
' Capture data.
' --------------------------------------------------------------------
Private Shared Sub CaptureData(ByVal myScope As IIviScope)

  ' Configure the trigger type and holdoff. Type can be edge,
  ' pulse width, glitch, TV, or ACLine (runt and immediate
  ' not supported in InfiniiVision oscilloscopes).
  ' ------------------------------------------------------------

  ' Set up edge trigger.
  Dim dHoldoff As Double = 0.00000006R   ' 60ns to 10s.
  myScope.Trigger.Configure( _
      IviScopeTriggerTypeEnum.IviScopeTriggerEdge, dHoldoff)

  myScope.Trigger.Edge.Configure("Channel1", 2.5, _
      IviScopeTriggerSlopeEnum.IviScopeTriggerSlopePositive)
  Console.WriteLine("Trigger edge source: " & _
      myScope.Trigger.Source.ToString())
  Console.WriteLine("Trigger edge level: " & _
      myScope.Trigger.Level.ToString())
  Console.WriteLine("Trigger edge slope: " & _
      myScope.Trigger.Edge.Slope.ToString())

  ' Trigger coupling.
  myScope.Trigger.Coupling = _
      IviScopeTriggerCouplingEnum.IviScopeTriggerCouplingDC
  Console.WriteLine("Trigger coupling: " & _
      myScope.Trigger.Coupling.ToString())

  ' Trigger modifier (Auto vs. Normal).
  myScope.Trigger.Modifier = _
      IviScopeTriggerModifierEnum.IviScopeTriggerModifierNone
  Console.WriteLine("Trigger modifier: " & _
      myScope.Trigger.Modifier.ToString())

  ' Set vertical range and offset.
  ' --------------------------------------------------------------------
  Dim myScopeCh As IIviScopeChannel
  myScopeCh = myScope.Channels.Item("Channel1")

  ' Channel coupling.
  myScopeCh.Coupling = _
      IviScopeVerticalCouplingEnum.IviScopeVerticalCouplingDC
  Console.WriteLine("Ch1 coupling: " & _
      myScopeCh.Coupling.ToString())

  ' Make sure channel is enabled.
```

```
myScopeCh.Enabled = True
Console.WriteLine("Ch1 enabled: " & _
    myScopeCh.Enabled.ToString())

' Vertical range.
myScopeCh.Range = 8.0R
Console.WriteLine("Ch1 vertical range: " & _
    myScopeCh.Range.ToString())

' Vertical offset.
myScopeCh.Offset = 2.5
Console.WriteLine("Ch1 vertical offset: " & _
    myScopeCh.Offset.ToString())

' Set horizontal scale and offset.
' -------------------------------------------------------------
Dim dRange As Double = 0.0000001R
' Horiz range.
Dim nSamples As Integer = 1000
' Number of points in waveform.
Dim dOffset As Double = -0.00000005R
' Horiz position.
myScope.Acquisition.ConfigureRecord(dRange, nSamples, dOffset)
Console.WriteLine("Record length (points): {0:D}", _
    myScope.Acquisition.RecordLength)
Console.WriteLine("Record start time: {0:E}", _
    myScope.Acquisition.StartTime)
Console.WriteLine("Time of record: {0:E}", _
    myScope.Acquisition.TimePerRecord)

' Set the acquisition type. Type can be Normal, PeakDetect,
' HiRes, or Average (Envelope not supported in InfiniiVision
' oscilloscopes).
' -------------------------------------------------------------

' Interpolation setting.
myScope.Acquisition.Interpolation = _
    IviScopeInterpolationEnum.IviScopeInterpolationSineX
Console.WriteLine("Interpolation: " & _
    myScope.Acquisition.Interpolation.ToString())

' Averaging.
myScope.Acquisition.Type = _
    IviScopeAcquisitionTypeEnum.IviScopeAcquisitionTypeAverage
Console.WriteLine("Acquisition type: " & _
    myScope.Acquisition.Type.ToString())

myScope.Acquisition.NumberOfAverages = 1024
Console.WriteLine("Number of averages: {0}", _
    myScope.Acquisition.NumberOfAverages)

' Or, configure by performing Auto-Scale.
' -------------------------------------------------------------
'myScope.Measurements.AutoSetup();

' Acquire data.
' -------------------------------------------------------------
```

```
  myScope.Measurements.Initiate()
  Console.WriteLine("Measurements status: " & _
      myScope.Measurements.Status().ToString())
  Console.WriteLine("Sample mode: " & _
      myScope.Acquisition.SampleMode.ToString())
  Console.WriteLine("Points per second: {0:E}", _
      myScope.Acquisition.SampleRate)
End Sub

' ====================================================================
' Analyze the captured data.
' --------------------------------------------------------------------
Private Shared Sub AnalyzeData(ByVal myScope As IIviScope)
  ' Set up measurements hashtable.
  Dim htMeasurements As New Hashtable()
  htMeasurements.Add( _
    IviScopeMeasurementEnum.IviScopeMeasurementRiseTime, _
    "rise time")
  htMeasurements.Add( _
    IviScopeMeasurementEnum.IviScopeMeasurementVoltagePeakToPeak, _
    "voltage pk-pk")

  ' Measurements and waveforms are available for each channel.
  For i As Integer = 1 To myScope.Measurements.Count
    ' measName is the name of a channel.
    Dim measName As String = _
        myScope.Measurements.Name(i).ToString()

    ' Operate on Measurement item.
    Dim myScopeCh As IIviScopeChannel
    myScopeCh = myScope.Channels.Item(measName)
    Dim myScopeMeas As IIviScopeMeasurement
    myScopeMeas = myScope.Measurements.Item(measName)

    If myScopeCh.Enabled Then
      ' Get measurement values.
      ' ------------------------------------------------------------
      Dim cMeasurements As ICollection = htMeasurements.Keys
      For Each eMeas As IviScopeMeasurementEnum In cMeasurements
        ' Display measurement.
        Dim dValue As Double = 0.0R
        myScopeMeas.FetchWaveformMeasurement(eMeas, dValue)
        Console.WriteLine("{0} {1}: {2:E}", _
            measName, htMeasurements(eMeas), dValue)
      Next

      ' Get waveform data.
      ' ------------------------------------------------------------
      Dim waveformData As Double() = New Double(999) {}
      Dim dInitialX As Double = 0.0R
      Dim dXIncrement As Double = 0.0R

      myScopeMeas.FetchWaveform(waveformData, dInitialX, _
          dXIncrement)
      Console.WriteLine("Data points fetched: {0}", _
          waveformData.Length)
```

```
            ' Set up output file:
            Dim strPath As String
            strPath = "c:\scope\data\waveform_data.csv"
            If File.Exists(strPath) Then
              File.Delete(strPath)
            End If

            ' Open file for output.
            Dim writer As StreamWriter = File.CreateText(strPath)

            ' Output waveform data in CSV format.
            For j As Integer = 0 To waveformData.Length - 1
              writer.WriteLine("{0:E}, {1:F6}", _
                  (dInitialX + (CSng(j) * dXIncrement)), _
                  waveformData(j))
            Next

            ' Close output file.
            writer.Close()
            Console.WriteLine("Waveform data written to {0}", strPath)
          End If
        Next
      End Sub

      ' ====================================================================
      ' Read instrument errors.
      ' --------------------------------------------------------------------
      Private Shared Sub _
          ReadInstrumentErrors(ByVal driver As IIviDriver)
        ' Read instrument error queue until its empty.
        Dim errCode As Integer = 0
        Dim errMsg As String = Nothing

        'Console.WriteLine();
        Do
          driver.Utility.ErrorQuery(errCode, errMsg)
          If errCode <> 0 Then
            Console.WriteLine("ErrorQuery: {0}, {1}", errCode, errMsg)
          End If
        Loop While errCode <> 0
      End Sub
    End Class
End Namespace
```

Visual Basic for Applications (VBA) Quick Start

To run this example in Visual Basic for Applications (VBA):

1. Start the application that provides Visual Basic for Applications (for example, Microsoft Excel).
2. Press ALT+F11 to launch the Visual Basic editor.
3. Reference the Agilent VISA COM library:
   a. Choose **Tools>References...** from the main menu.
   b. In the References dialog, select these libraries:
      - IviDriver 1.0 Type Library (for inherent capabilities)
      - IviScope 3.0 Type Library (for class-compliant interface)
      - IviSessionFactory 1.0 Type Library (facilitates interchangeable COM applications)
      - IVI Agilent546XX 1.3 Type Library (for instrument-specific interface)
   c. Click **OK**.
4. Choose **Insert>Module**.
5. Cut-and-paste the code that follows into the editor.
6. Run the program.

```
'
' IVI-COM Driver for InfiniiVision Oscilloscopes, Visual Basic
' ====================================================================

Option Explicit

Public strResult As String

'
' Main Program
' --------------------------------------------------------------------

Sub Main()

  Dim logicalName As String
  logicalName = "myInfiniiVisionScope"

  Dim factory As New IviSessionFactory
  Dim driver As IIviDriver
  Set driver = factory.CreateDriver(logicalName)

  ' Display driver identity - Initialize not required.
  DisplayDriverIdentity driver

  ' Set up IVI-defined initialization options.
  Dim standardInitOptions As String
  standardInitOptions = _
```

```
        "Cache=true, " & _
        "InterchangeCheck=false, " & _
        "QueryInstrStatus=false, " & _
        "RangeCheck=true, " & _
        "RecordCoercions=false, " & _
        "Simulate=false"

    ' Set up driver-specific initialization options.
    Dim driverSetupOptions As String
    driverSetupOptions = "DriverSetup= Model=Agilent546XX, Trace=false"

    driver.Initialize logicalName, False, False, _
        standardInitOptions & "," & driverSetupOptions

    ' Display instrument identity - Initialize required.
    DisplayInstrumentIdentity driver

    ' Exercise IIviScope class-compliant methods and properties.
    Dim myScope As IIviScope
    Set myScope = driver

    On Error GoTo ErrorHandler

    ' Initialize the oscilloscope to a known state.
    InitializeScope myScope

    ' Capture data.
    CaptureData myScope

    ' Analyze the captured data.
    AnalyzeData myScope

    ' Check instrument for errors.
    ReadInstrumentErrors driver

    ' Close driver if initialized.
    If driver.Initialized Then
      Debug.Print "Closing initialized driver..."
      driver.Close
    End If

Exit Sub

ErrorHandler:
    Debug.Print "ErrorHandler: " & Err.description
    If driver.Initialized Then driver.Close
    Exit Sub
End Sub

' ====================================================================
' Display driver identity properties (no initialization req'd).
' --------------------------------------------------------------------
Private Sub DisplayDriverIdentity(driver As IIviDriver)

    strResult = driver.Identity.identifier
    Debug.Print "Driver identifier: " & strResult
```

```
  strResult = driver.Identity.description
  Debug.Print "Driver description: " & strResult

  strResult = driver.Identity.revision
  Debug.Print "Driver revision: " & strResult

  strResult = driver.Identity.vendor
  Debug.Print "Driver version: " & strResult

End Sub

' ====================================================================
' Display instrument identity properties (initialization req'd).
' --------------------------------------------------------------------
Private Sub DisplayInstrumentIdentity(driver As IIviDriver)

  strResult = driver.Identity.InstrumentModel
  Debug.Print "Instrument model: " & strResult

  strResult = driver.Identity.InstrumentFirmwareRevision
  Debug.Print "Instrument firmware revision: " & strResult

  strResult = driver.Identity.InstrumentManufacturer
  Debug.Print "Instrument manufacturer: " & strResult

End Sub

' ====================================================================
' Initialize the oscilloscope to a known state.
' --------------------------------------------------------------------
Private Sub InitializeScope(myScope As IIviScope)

  myScope.Utility.Reset

End Sub

' ====================================================================
' Capture data.
' --------------------------------------------------------------------
Private Sub CaptureData(myScope As IIviScope)

  ' Configure the trigger type and holdoff. Type can be edge,
  ' pulse width, glitch, TV, or ACLine (runt and immediate
  ' not supported in InfiniiVision oscilloscopes).
  ' ------------------------------------------------------------------

  ' Set up edge trigger.
  Dim dHoldoff As Double
  dHoldoff = 0.00000006   ' 60ns to 10s.
  myScope.Trigger.Configure _
      IviScopeTriggerTypeEnum.IviScopeTriggerEdge, dHoldoff

  myScope.Trigger.Edge.Configure "Channel1", 2.5, _
      IviScopeTriggerSlopeEnum.IviScopeTriggerSlopePositive
  Debug.Print "Trigger edge source: " & myScope.Trigger.Source
  Debug.Print "Trigger edge level: " & myScope.Trigger.Level
  Debug.Print "Trigger edge slope: " & myScope.Trigger.Edge.Slope
```

```
' Trigger coupling.
myScope.Trigger.Coupling = _
    IviScopeTriggerCouplingEnum.IviScopeTriggerCouplingDC
Debug.Print "Trigger coupling: " & myScope.Trigger.Coupling

' Trigger modifier (Auto vs. Normal).
myScope.Trigger.Modifier = _
    IviScopeTriggerModifierEnum.IviScopeTriggerModifierNone
Debug.Print "Trigger modifier: " & myScope.Trigger.Modifier

' Set vertical range and offset.
' ----------------------------------------------------------------
Dim myScopeCh As IIviScopeChannel
Set myScopeCh = myScope.Channels.Item("Channel1")

' Channel coupling.
myScopeCh.Coupling = _
    IviScopeVerticalCouplingEnum.IviScopeVerticalCouplingDC
Debug.Print "Ch1 coupling: " & myScopeCh.Coupling

' Make sure channel is enabled.
myScopeCh.Enabled = True
Debug.Print "Ch1 enabled: " & myScopeCh.Enabled

' Vertical range.
myScopeCh.Range = 8#
Debug.Print "Ch1 vertical range: " & myScopeCh.Range

' Vertical offset.
myScopeCh.Offset = 2.5
Debug.Print "Ch1 vertical offset: " & myScopeCh.Offset

' Set horizontal scale and offset.
' ----------------------------------------------------------------
Dim dRange As Double
dRange = 0.0000001
' Horiz range.
Dim nSamples As Integer
nSamples = 1000
' Number of points in waveform.
Dim dOffset As Double
dOffset = -0.00000005

' Horiz position.
myScope.Acquisition.ConfigureRecord dRange, nSamples, dOffset
Debug.Print "Record length (points): " & _
    FormatNumber(myScope.Acquisition.RecordLength, 0)
Debug.Print "Record start time: " & _
    Format(myScope.Acquisition.StartTime, "Scientific")
Debug.Print "Time of record: " & _
    Format(myScope.Acquisition.TimePerRecord, "Scientific")

' Set the acquisition type. Type can be Normal, PeakDetect,
' HiRes, or Average (Envelope not supported in InfiniiVision
' oscilloscopes).
' ----------------------------------------------------------------
```

```
  ' Interpolation setting.
  myScope.Acquisition.Interpolation = _
      IviScopeInterpolationEnum.IviScopeInterpolationSineX
  Debug.Print "Interpolation: " & myScope.Acquisition.Interpolation

  ' Averaging.
  myScope.Acquisition.Type = _
      IviScopeAcquisitionTypeEnum.IviScopeAcquisitionTypeAverage
  Debug.Print "Acquisition type: " & myScope.Acquisition.Type

  myScope.Acquisition.NumberOfAverages = 1024
  Debug.Print "Number of averages: " & _
      FormatNumber(myScope.Acquisition.NumberOfAverages, 0)

  ' Or, configure by performing Auto-Scale.
  ' -------------------------------------------------------------------
  'myScope.Measurements.AutoSetup();

  ' Acquire data.
  ' -------------------------------------------------------------------
  myScope.Measurements.Initiate
  Debug.Print "Measurements status: " & myScope.Measurements.Status()
  Debug.Print "Sample mode: " & myScope.Acquisition.SampleMode
  Debug.Print "Points per second: " & _
      Format(myScope.Acquisition.SampleRate, "Scientific")

End Sub

' ====================================================================
' Analyze the captured data.
' -------------------------------------------------------------------
Private Sub AnalyzeData(myScope As IIviScope)

  ' Measurements and waveforms are available for each channel.
  Dim i As Integer
  For i = 1 To myScope.Measurements.Count
    ' measName is the name of a channel.
    Dim measName As String
    measName = myScope.Measurements.Name(i)

    ' Operate on Measurement item.
    Dim myScopeCh As IIviScopeChannel
    Set myScopeCh = myScope.Channels.Item(measName)
    Dim myScopeMeas As IIviScopeMeasurement
    Set myScopeMeas = myScope.Measurements.Item(measName)

    If myScopeCh.Enabled Then

      ' Get measurement values.
      ' -----------------------------------------------------------------
      Dim dValue As Double
      dValue = 0#

      myScopeMeas.FetchWaveformMeasurement _
          IviScopeMeasurementRiseTime, dValue
      Debug.Print measName & " rise time: " & _
```

```
            Format(dValue, "Scientific")

        myScopeMeas.FetchWaveformMeasurement _
            IviScopeMeasurementVoltagePeakToPeak, dValue
        Debug.Print measName & " voltage pk-pk: " & _
            Format(dValue, "Scientific")

        ' Get waveform data.
        ' --------------------------------------------------------------------
        Dim waveformData() As Double
        Dim dInitialX As Double
        Dim dXIncrement As Double
        Dim j As Integer

        'Dim waveformData As Double() = New Double(999) {}
        'Dim dInitialX As Double
        'dInitialX = 0#
        'Dim dXIncrement As Double
        'dXIncrement = 0#

        myScopeMeas.FetchWaveform waveformData, dInitialX, dXIncrement
        Debug.Print "Data points fetched: " & UBound(waveformData) + 1

        ' Set up output file:
        Dim strPath As String
        strPath = "c:\scope\data\waveform_data.csv"

        ' Open file for output.
        Dim hFile As Long
        hFile = FreeFile
        Open strPath For Output Access Write Lock Write As hFile

        ' Output waveform data in CSV format.
        For j = 0 To UBound(waveformData)
          ' Write time value, voltage value.
          Print #hFile, _
              FormatNumber(dInitialX + (CSng(j) * dXIncrement), 12) + _
              ", " + FormatNumber(waveformData(j), 6)
        Next j

        ' Close output file.
        Close hFile    ' Close file.

        Debug.Print "Waveform data written to " & strPath

    End If
  Next

End Sub

' ====================================================================
' Read instrument errors.
' --------------------------------------------------------------------
Private Sub ReadInstrumentErrors(driver As IIviDriver)

  ' Read instrument error queue until it is empty.
  Dim errorNum As Long
```

```
  Dim errorMsg As String
  errorNum = -1

  While errorNum <> 0
    driver.Utility.ErrorQuery errorNum, errorMsg
    Debug.Print "ErrorQuery: " & CStr(errorNum) & ", " & errorMsg
  Wend

End Sub
```

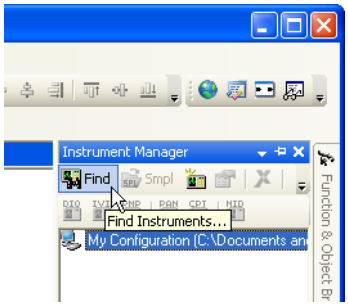
Agilent VEE Pro Quick Start

1 Start Agilent VEE Pro.

2 Find instruments:

a From the Agilent VEE Pro window's main menu, choose **I/O>Instrument Manager...**.

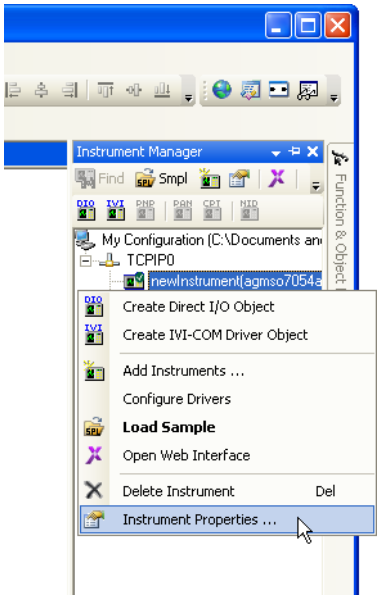
b In the Instrument Manager pane, click **Find**.



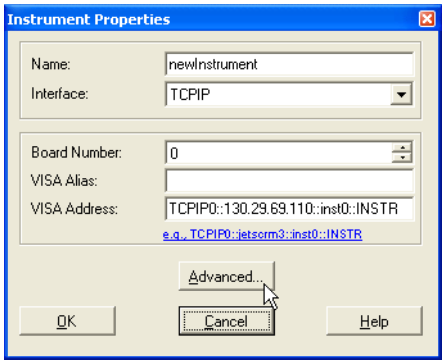
c For each Identify Instrument dialog that appears, click **OK**.

3 Edit your oscilloscope's instrument properties:

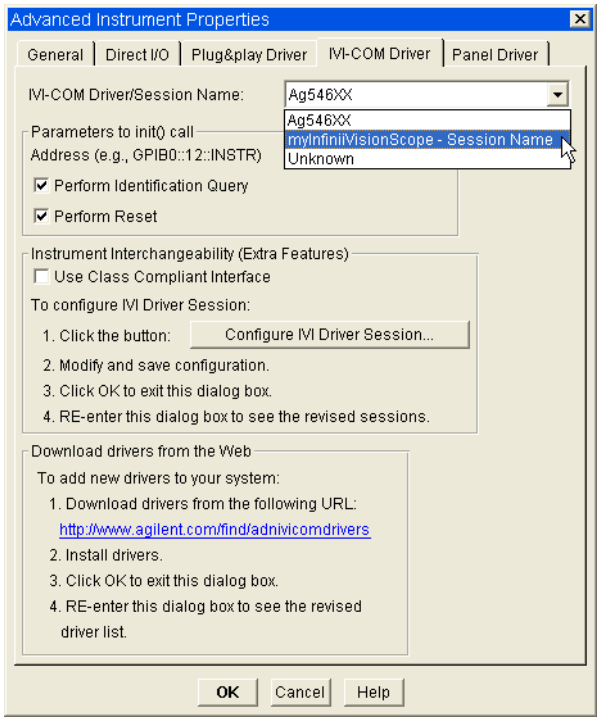
a Richt-click your oscilloscope instrument in the Instrument Manager pane and choose **Instrument Properties...**.



b In the Instrument Properties dialog, click **Advanced...**.



c In the Advanced Instrument Properties dialog's IVI-COM Driver tab, select the IVI-COM session name.



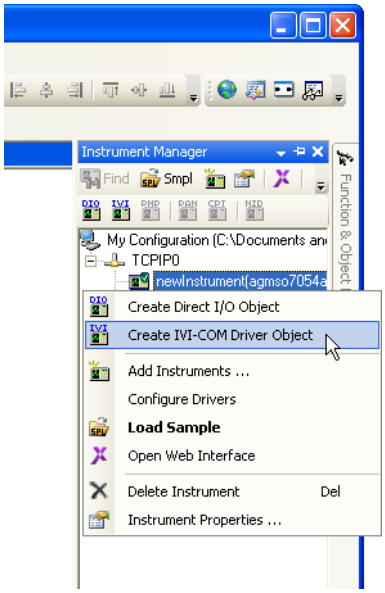
d If you would like to use the class-compliant interface, check **Use Class Compliant Interface**.

e Click **OK** to close the Advanced Instrument Properties dialog.

f Click **OK** to close the Instrument Properties dialog.

4 Create the IVI-COM driver object::

a Richt-click your oscilloscope instrument in the Instrument Manager pane and choose **Create IVI-COM Driver Object**.

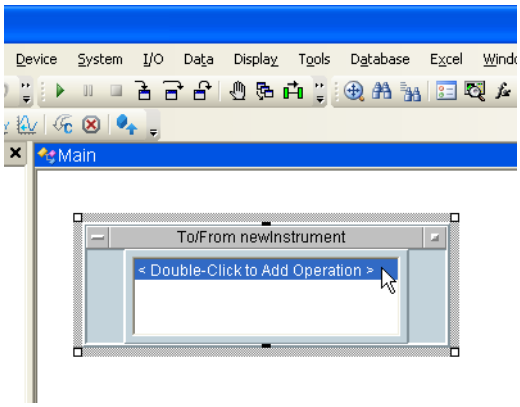


You will see an outline for the driver object box in the Main pane.

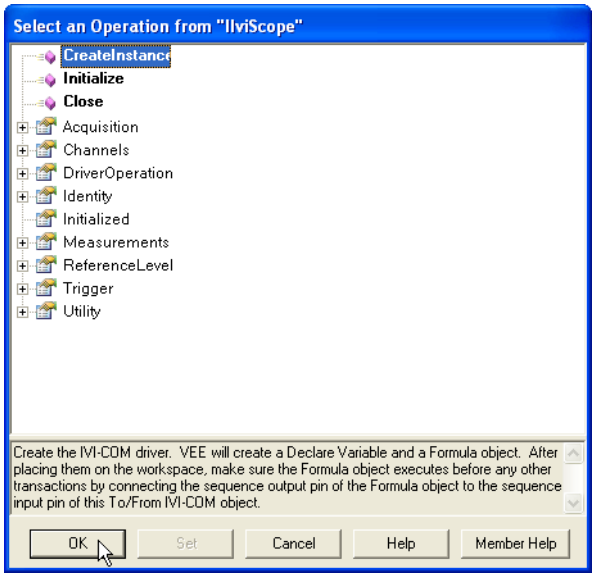
b Click anywhere in the Main pane to locate the driver object.

5 Create an instance of the driver, and add Initialize and Close methods:

a In the driver object box, double-click **< Double-Click to Add Operation >**.



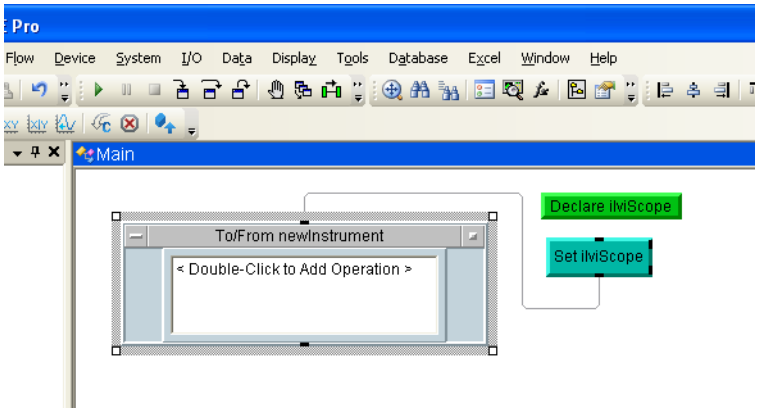
b In the Select an Operation dialog, select **CreateInstance**; then, click **OK**.



You will see an outline for the driver instance boxes in the Main pane.

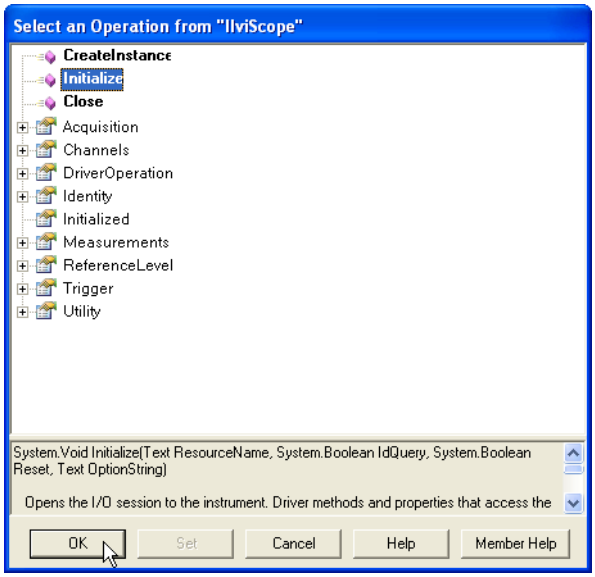
Click anywhere in the Main pane to locate the driver instance boxes.

The Declare box and the Set box create an instance of the driver.



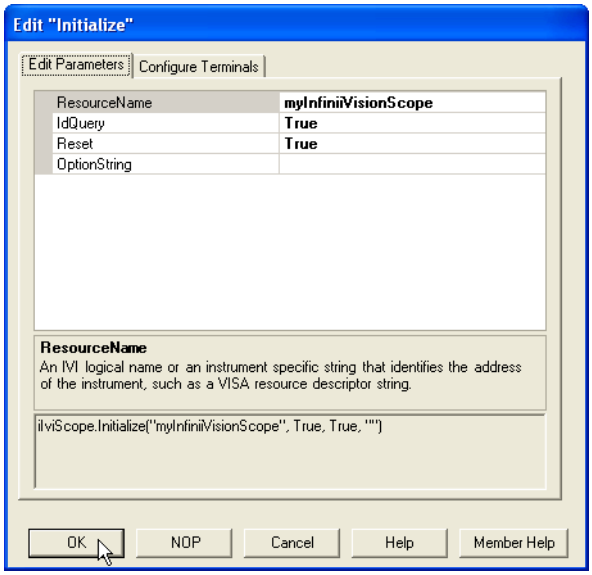
c Double-click **< Double-Click to Add Operation >**.

d In the Select an Operation dialog, select **Initialize**; then, click **OK**.



e In the Edit "Initialize" dialog's Edit Parameters tab, enter the desired parameter values.

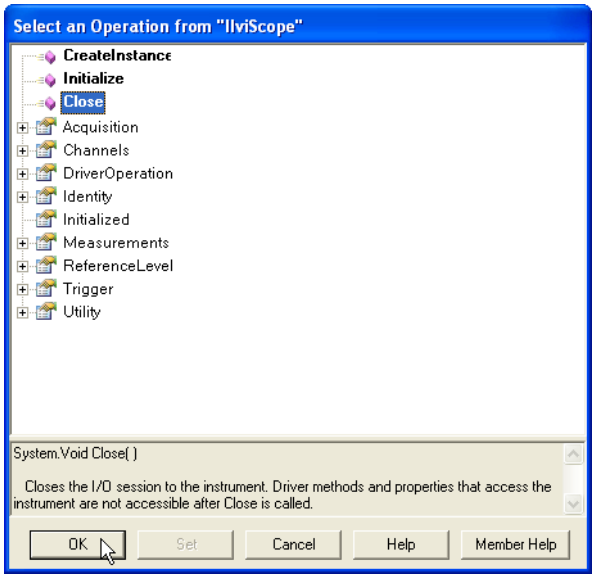
Select the logical name you added to the IVI Configuration Store as the ResourceName parameter.



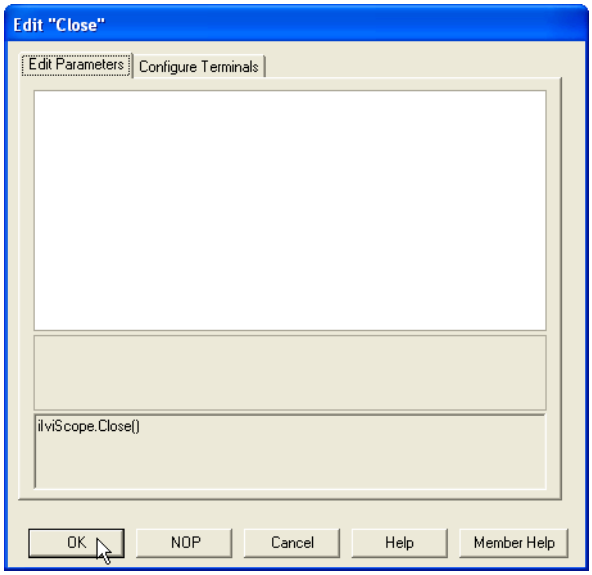
Then, click **OK**.

f Double-click **< Double-Click to Add Operation >**.

g In the Select an Operation dialog, select **Close**; then, click **OK**.



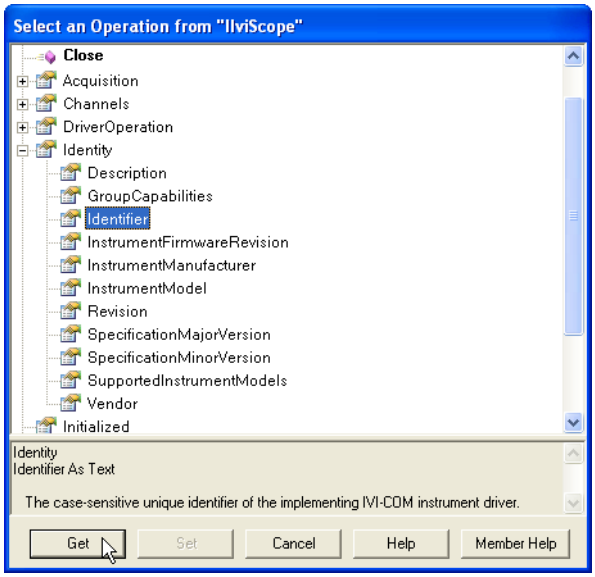
h In the Edit "Close" dialog's Edit Parameters tab, there are no parameters to edit, so click **OK**.



6 Get and display the driver identifier property:

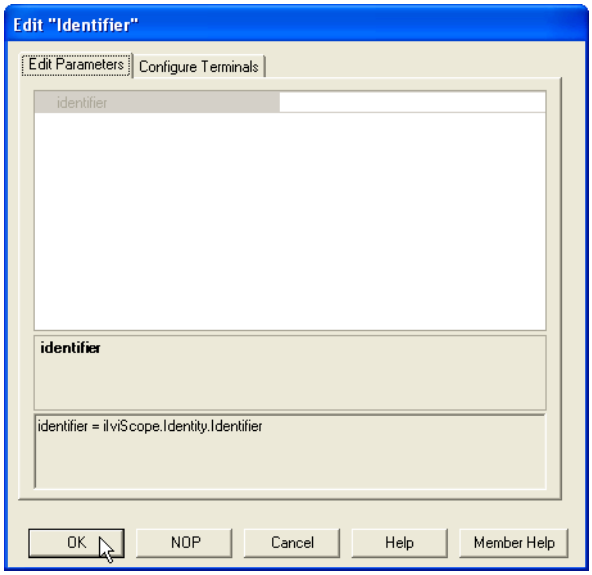
a Double-click **< Double-Click to Add Operation >**.

b In the Select an Operation dialog, expand the **Identity** property, select **Identifier**; then, click **Get**.



c In the Edit "Identifier" dialog's Edit Parameters tab, there are no parameters to edit.

Notice, however, that the property is returned to an "identifier" parameter.



Click **OK**.

Notice that the "identifier" parameter appears an an output in the "To/From newInstrument" driver object box.

d With the get property operation selected in the "To/From newInstrument" driver object box, press Ctrl-Up to move it up in the list of operations.

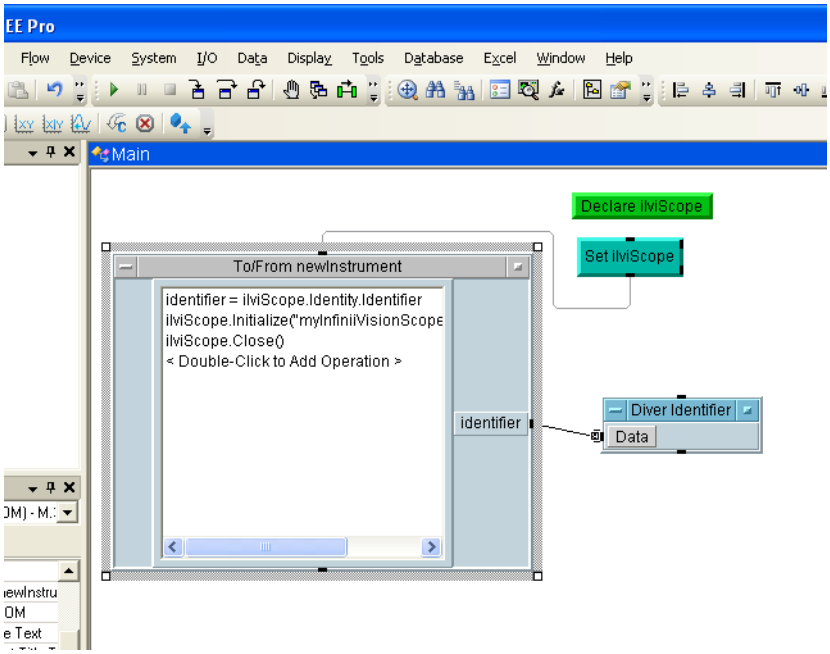
You can get the driver identifier property before the driver is initialized, so move it to the top of the list.

e From the Agilent VEE Pro window's main menu, choose **Display>AlphaNumeric**.

f Click in the Main pane to locate the alphanumeric display box.

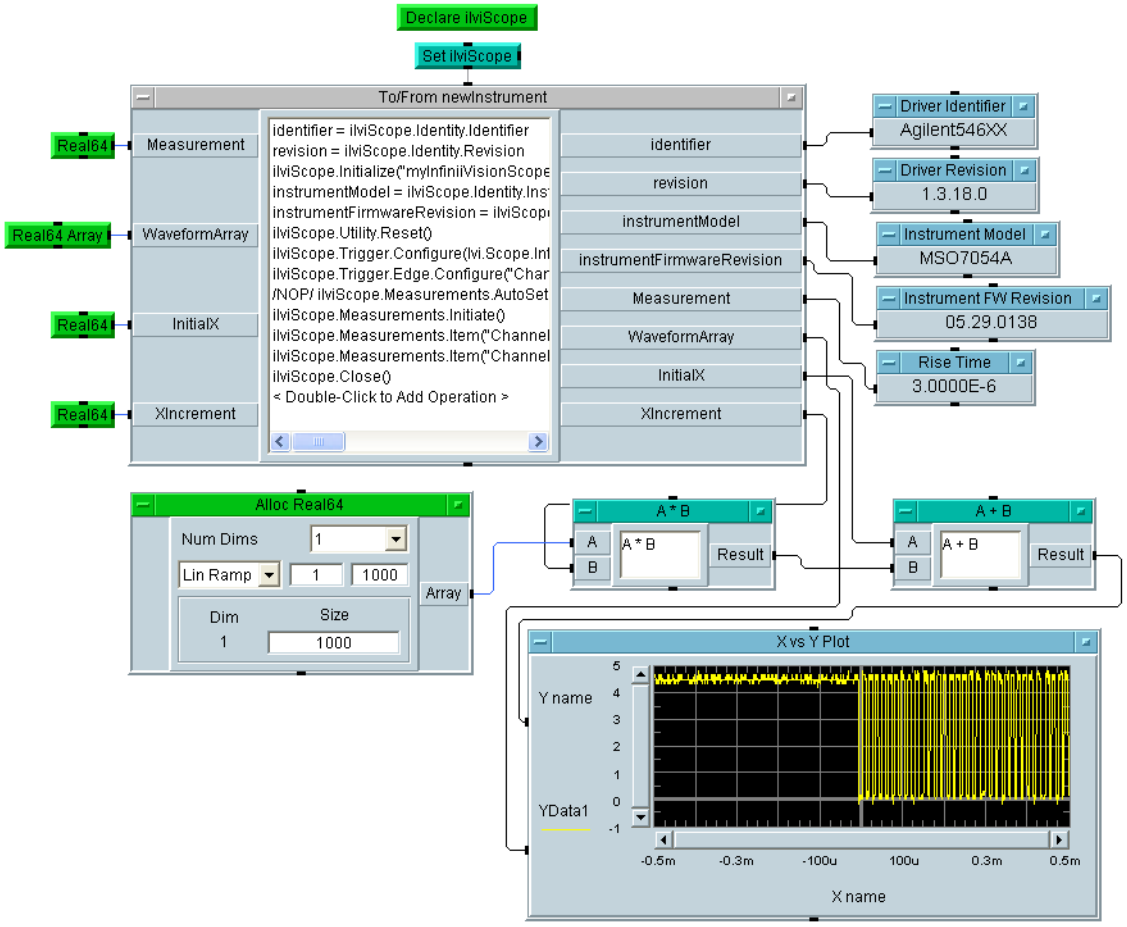
g With the AlphaNumeric display box selected, change its title in the Properties pane to "Driver Identity".

h Now, draw a line from the "identifier" output node to the alphanumeric display input node.



7 Click the green-triangle Run button to run the program.

8 You can continue adding operations to configure the oscilloscope, acquire data, make measurements, and get waveform data, as shown below.



In the example above, note that:

- When editing operations, you can add a NOP (no-operation) to keep it from being executed.
- For the X vs Y plot, the InitialX and XIncrement parameters returned from the FetchWaveform method are used to calculate X axis values for the 1000 points of waveform data returned.

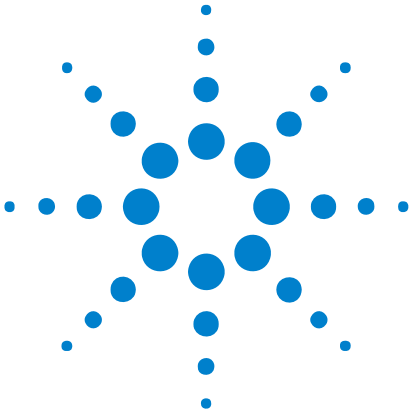
For More Examples

For more examples of programming using the Agilent546XX IVI-COM driver, see:

- The Agilent546XX IVI driver examples:

  From the Windows Start menu, choose **Start>All Programs>Agilent IVI Drivers>Agilent546XX>Examples**.

  This opens a folder that contains a Read Me file and sub-folders that contain examples for the different supported programming environments.



4 Agilent 546XX IVI-COM Driver Notes

Notes on the Driver Documentation

The Agilent546XX IVI driver online documentation has many examples that refer to "Dmm", "IviDmm", "Foo", or "Company.Foo". These are generic examples using the driver for a digital multimeter or a fictitious company's "Foo" driver. When you see these, substitute "Scope", "IviScope", "Agilent546XX", or "Agilent.Agilent546XX".

Class-Compliant Driver Limitations

The Agilent InfiniiVision oscilloscopes do not support these are parts of the IviScope class-compliant interface:

- Class-compliant Runt trigger and Immediate trigger types are not supported.
- Pulse width trigger or glitch trigger polarity cannot be "either" (that is, either positive or negative).
- AcLine trigger slope cannot be "either" (that is, either positive or negative).
- The AutoLevel trigger modifier is not supported.
- The Envelope acquisition type is not supported.
- The Ground channel coupling is not supported.
- The Linear interpolation for acquisition is not supported.

Of course, the Agilent InfiniiVision oscilloscopes do not support these parts of the Agilent546XX interface either.

Class-Compliant and Instrument-Specific Driver Differences

The Agilent546XX instrument-specific driver interface gives you these added capabilities:

- Access to the digital channels of a mixed-signal oscilloscope (MSO). Use "UserDigitalChannel0" through "UserDigitalChannel15" when specifying digital channels.
- You can use "UserChannel1" , etc., in addition to "Channel1", etc., for analog channels.
- The instrument-specific interface's IAgilent546XX.ConfigurableMeasurement property lets you make time-at-edge, time-at-value, and value-at-time measurements.
- The instrument-specific interface's IAgilent546XX.Cursors property lets you position the X and Y cursors and set the analog input channel source.
- The instrument-specific interface's IAgilent546XX.DigitalChannels property lets you enable and label digital channels, set their threshold voltages, and get captured digital channel data.
- The instrument-specific interface's IAgilent546XX.Display property lets you clear the display, show labels for enabled channels, and set waveform persistence options.
- The instrument-specific interface's IAgilent546XX.External property lets you set options for the external trigger input.
- The instrument-specific interface's IAgilent546XX.MultiWaveformMeasurement property lets you set options for the two-channel delay and phase measurements, and it lets you read and reset digital channel activity.
- The instrument-specific interface's IAgilent546XX.Status property lets you perform operations on the oscilloscope's status reporting registers.
- The instrument-specific interface's IAgilent546XX.System property lets you:
  - Get, put, save, and recall "states", that is, oscilloscope setups.
  - Set and get the oscilloscope's date and time.
  - Disable and enable (that is, lock and unlock) the oscilloscope's front panel controls.
  - Get the oscilloscope's screen bitmap image.
  - Get the oscilloscope's serial number.
  - Set the timeout used for I/O operations.
  - Wait for oscilloscope operations to complete.

  - Access the IFormattedIO488 interface and use the oscilloscope's built-in SCPI commands. The examples provided with the Agilent546XX driver show you how to access the IFormattedIO488 interface.

- The instrument-specific interface's IAgilent546XXTrigger.Pattern property and IAgilent546XXTrigger.State property let you set up pattern triggers.

4 Agilent 546XX IVI-COM Driver Notes

5 For More Information

For more information on using IVI-COM drivers, see:

- The Agilent546XX IVI driver Readme file:

  From the Windows Start menu, choose **Start>All Programs>Agilent IVI Drivers>Agilent546XX>Readme**.

  This opens a text file that contains read me first and revision information for the Agilent546XX IVI-COM driver.

- The Agilent546XX IVI driver online Documentation:

  From the Windows Start menu, choose **Start>All Programs>Agilent IVI Drivers>Agilent546XX>Documentation**.

  This opens a Windows HTML Help file that is the documentation for the Agilent546XX IVI-COM driver. It contains additional getting started and API reference information. You can find additional notes about this documentation at "Notes on the Driver Documentation" on page 66.

- The getting started guides at the "www.ivifoundation.org" web site.

5 For More Information

Index

Index