Keysight M9217A PXIe 2-CH Digitizer, Isolated, ±256 V, 20 MSa/s, 16-Bit

Guide to write and compile source code to produce an Assembly .exe file

Programming Guide

KEYSIGHT TECHNOLOGIES

Notices

Manual Part Number

M9217-90006

Edition

Edition 1, October 1, 2015

Printed in Malaysia

Keysight Technologies
1400 Fountaingrove Parkway
Santa Rosa, CA 95403

Sales and Technical Support

To contact Keysight for sales and technical support, refer to the "support" links on the following Keysight Web resources:

- www.keysight.com/find/m9217a (product-specific information and support, software and documentation updates)
- www.keysight.com/find/assist (worldwide contact information for repair and service)

Information on preventing damage to your Keysight equipment can be found at www.keysight.com/find/tips.

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 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 WILL 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

U.S. Government Restricted Rights. Software and technical data rights granted to the federal government include only those rights customarily provided to end user customers. Keysight provides this customary commercial license in Software and technical data pursuant to FAR 12.211 (Technical Data) and 12.212 (Computer Software) and, for the Department of Defense, DFARS 252.227-7015 (Technical Data - Commercial Items) and DFARS 227.7202-3 (Rights in Commercial Computer Software or Computer Software Documentation).

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.

WARNING

IF THIS PRODUCT IS NOT USED AS SPECIFIED, THE PROTECTION PROVIDED BY THE EQUIPMENT COULD BE IMPAIRED. THIS PRODUCT MUST BE USED IN A NORMAL CONDITION (IN WHICH ALL MEANS FOR PROTECTION ARE INTACT) ONLY.

The types of product users are:

- Responsible body is the individual or group responsible for the use and maintenance of equipment, for ensuring that the equipment is operated within its specifications and operating limits, and for ensuring that operators are adequately trained.
- Operators use the product for its intended function. They must be trained in electrical safety procedures and proper use of the instrument. They must be protected from electric shock and contact with hazardous live circuits.
- Maintenance personnel perform routine procedures on the product to keep it operating properly (for example, setting the line voltage or replacing consumable materials). Maintenance procedures are described in the user documentation. The procedures explicitly state if the operator may perform them. Otherwise, they should be performed only by service personnel.
- Service personnel are trained to work on live circuits, perform safe installations, and repair products. Only properly trained service personnel may perform installation and service procedures.

Exercise extreme caution when a shock hazard is present. Lethal voltage may be present on cable connector jacks or test fixtures. The American National Standards Institute (ANSI) states that a shock hazard exists when voltage levels greater than 30 V RMS, 42.4 V peak, or 60 VDC are present.

A good safety practice is to expect that hazardous voltage is present in any unknown circuit before measuring.

Operators of this product must be protected from electric shock at all times. The responsible body must ensure that operators are prevented access and/or insulated from every connection point. In some cases, connections must be exposed to potential human contact. Product operators in these circumstances must be trained to protect themselves from the risk of electric shock.

Before operating an instrument, ensure that the line cord is connected to a properly-grounded power receptacle. Inspect the connecting cables, test leads, and jumpers for possible wear, cracks, or breaks before each use.

When installing equipment where access to the main power cord is restricted, such as rack mounting, a separate main input power disconnect device must be provided in closed proximity to the equipment and within easy reach of the operator.

For maximum safety, do not touch the product, test cables, or any other instruments while power is applied to the circuit under test. ALWAYS remove power from the entire test system and discharge any capacitors before: connecting or disconnecting cables or jumpers, installing or removing switching cards, or making internal changes, such as installing or removing jumpers.

Do not touch any object that could provide a current path to the common side of the circuit under test or power line (earth) ground. Always make measurements with dry hands while standing on a dry, insulated surface capable of withstanding the voltage being measured.

The instrument and accessories must be used in accordance with its specifications and operating instructions, or the safety of the equipment may be impaired.

CAUTION

To maintain protection from electric shock and fire, replacement components in mains circuits - including the power transformer, test leads, and input jacks - must be purchased from Keysight. Standard fuses with applicable national safety approvals may be used if the rating and type are the same. Other components that are not safety-related may be purchased from other suppliers as long as they are equivalent to the original component (note that selected parts should be purchased only through Keysight to maintain accuracy and functionality of the product). If you are unsure about the applicability of a replacement component, call a Keysight office for information.

- Do not exceed the maximum signal levels of the instruments and accessories, as defined in the specifications and operating information, and as shown on the instrument or test fixture panels, or switching card.
- Chassis connections must only be used as shield connections for measuring circuits, NOT as safety earth ground connections.
- If you are using a test fixture, keep the lid closed while power is applied to the device under test. Safe operation requires the use of a lid interlock.
- Instrumentation and accessories shall not be connected to humans.

WARNING

No operator serviceable parts inside. Refer servicing to qualified personnel. To prevent electrical shock do not remove covers.

Cleaning Precautions

WARNING

To prevent electrical shock, disconnect the Keysight Technologies instrument from mains before cleaning. Use a dry cloth or one slightly dampened with water to clean the external case parts. Do not attempt to clean internally. To clean the connectors, use alcohol in a well-ventilated area. Allow all residual alcohol moisture to evaporate, and the fumes to dissipate prior to energizing the instrument.

Side Panel Symbols

	The CSA mark is a registered trademark of the CSA International.
	The CE marking is the legal required labeling for several EU Directives of the European Union. The CE marking shows that the product complies with all relevant European Legal Directives.
	The RCM mark is a Compliance Mark according to the ACMA Labelling Requirement.
	The KC mark shows that the product complies with the relevant Korean Compulsory Certification.
	This symbol indicates product compliance with the Canadian Interference-Causing Equipment Standard (ICES-001). It also identifies the product is an Industrial Scientific and Medical Group 1 Class A product (CISPR 11, Clause 4).
	This symbol indicates the time period during which no hazardous or toxic substance elements are expected to leak or deteriorate during normal use. Forty years is the expected useful life of the product.

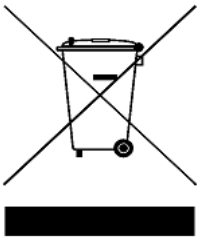
Waste Electrical and Electronic Equipment (WEEE) Directive 2002/96/EC

This instrument complies with the WEEE Directive (2002/96/EC) marking requirement. This affixed product label indicates that you must not discard this electrical or electronic product in domestic household waste.

Product category

With reference to the equipment types in the WEEE directive Annex 1, this instrument is classified as a “Monitoring and Control Instrument” product.

The affixed product label is as shown below.



Do not dispose in domestic household waste.

To return this unwanted instrument, contact your nearest Keysight Service Center, or visit www.keysight.com/environment/product for more information.

Table of Contents

THIS PAGE HAS BEEN INTENTIONALLY LEFT BLANK.

List of Figures

THIS PAGE HAS BEEN INTENTIONALLY LEFT BLANK.

List of Tables

THIS PAGE HAS BEEN INTENTIONALLY LEFT BLANK.

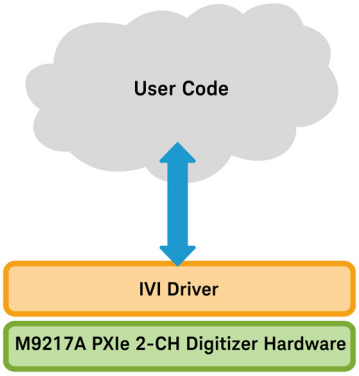
Keysight M9217A
PXIe 2-CH Digitizer, Isolated, ±256 V, 20 MSa/s, 16-Bit
Programming Guide

What You Will Learn in this Programming Guide

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 M9217A PXIe 2-CH Digitizer



IVI-COM Console Applications that are covered in this programming guide are used to perform the basic functionality of the M9217A PXIe 2-CH Digitizer.

The following basic functionality tests are covered:

Related websites

- Keysight Technologies PXI and AXIe Modular Products (http://www.keysight.com/find/modular)
  - M9217A PXIe 2-CH Digitizer (http://www.keysight.com/find/m9217a)
- Keysight Technologies (http://www.keysight.com/)
  - IVI Drivers and components downloads (http://www.keysight.com/find/ivi)

KEYSIGHT TECHNOLOGIES

- Keysight I/O Libraries Suite (http://www.keysight.com/find/iosuite)
- GPIB, USB, and Instrument Control Products (http://www.keysight.com/find/io)
- Keysighy VEE Pro (http://www.keysight.com/find/vee)
- Technical Support, Manuals, and Downloads (http://www.keysight.com/find/support)
- Contact Keysight Test and Measurement (http://www.keysight.com/find/contactus)

- IVI Foundation (http://www.ivifoundation.org/) - Usage Guides, Specifications, Shared Components Downloads
- MSDN Online (http://msdn.microsoft.com/)

Related documentation

To access documentation related to the IVI Driver, use one of the following:

Document	Link
Startup Guide[a] Includes procedures to help you to unpack, inspect, install (software and hardware), perform instrument connections, verify operability, and troubleshoot your product. Also includes an annotated block diagram.	C:\Program Files\Keysight\M9217\Help
Data Sheet[a] In addition to a detailed product introduction, the data sheet supplies full product specifications.	C:\Program Files\Keysight\M9217\Help
LabVIEW Driver Reference (Online Help System) Provides detailed documentation of the LabVIEW G Driver API functions.	C:\Program Files\Keysight\M9217\LabVIEW Driver Help

[a] If these links do not work, you can find these items at:
C:\Program Files\Keysight\M9217 or C:\Program Files(x86)\Keysight\M9217

Understanding the overall process flow

- Write source code using Microsoft Visual Studio 2010 with .NET Visual C# running on Windows 7.
- Compile source code using the .NET Framework Library.
- 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**).

Before Programming, Install Hardware, Software, and Licenses

1. Install Microsoft Visual Studio 2010 with .NET Visual C# running on Windows 7.

   The following steps are defined in the Keysight M9217A PXIe 2-CH Digitizer, Isolated, ±256 V, 20 MSa/s, 16-Bit Startup Guide, M9217-90001. However, these steps are repeated here and must be completed before you can programmatically control the M9217A PXIe 2-CH Digitizer hardware with these IVI Drivers.

2. Unpack and inspect all hardware.
3. Verify the shipment contents.
4. Install the software. Note the following order when installing software!

   (If you run the installation .exe, all of these are installed automatically.)

   - Install Keysight IO Libraries Suite (IOLS), Version 17.1.19313.5 or newer; this installation includes Keysight Connections Expert.
   - Install the M9217A PXIe 2-CH Digitizer driver software, Version 1.0.0.0 or newer. 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
   - http://www.keysight.com/find/ivi - download installers for Keysight IVI-COM drivers
   - http://www.keysight.com/find/m9217a

5. Install the hardware modules and make cable connections.
6. Verify operation of the modules (or the system that the modules create).

Once the software and hardware are installed and verification of operation has been performed, the M9217A is ready to be programmatically controlled.

Understanding the Application Programming Interfaces (API) for the M9217A PXIe 2-CH Digitizer

The following IVI Driver terminology may be used throughout this programming guide.

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

Currently, there are 13 IVI Instrument Classes defined by the IVI Foundation. The M9217A PXIe 2-CH Digitizer belongs to the IVI Digitizer Class.

IVI Instrument Classes (defined by the IVI Foundation)

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

IVI Compliant or IVI Class Compliant

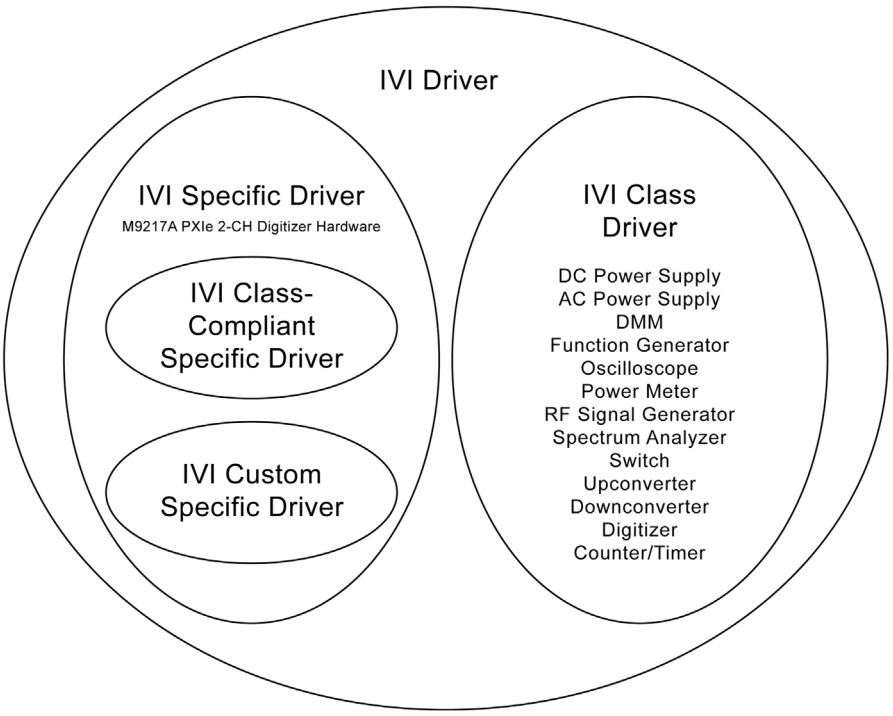
The M9217A PXIe 2-CH Digitizer are IVI Compliant and IVI Class Compliant; it belongs to IVI Digitizer Class defined by the IVI Foundation.

- **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 Specific Driver**
  - Is an IVI Driver that is written for a particular instrument such as the M9217A PXIe 2-CH Digitizer
- **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 Class-Compliant Specific Driver**
  - IVI Specific Driver that complies with one (or more) of the IVI defined class specifications
  - Used when hardware independence is desired
- **IVI Custom Specific Driver**
  - IVI Specific Driver that is not compliant with any one of the 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 IIviDriver interfaces.

- **Class-Compliant Hierarchy** - M9217A PXIe 2-CH Digitizer is compliant to IVI Digitizer Class. In its IVI Driver, there is IIviDigitizer root in Class-Compliant Hierarchy (where IIviDigitizer is the IVI Digitizer class-compliant root interface).

  IVI Digitizer Class specification divides the digitizer capabilities into a base capability group and multiple extension capability groups. The following table shows the supported group capabilities in the driver.

Table 1 Supported IviDigitizer group

Group name	Description	Supported
IviDigitizerBase	Base Capabilities of the IviDigitizer specification. This group includes the capability to acquire waveforms using edge triggering.	Yes
IviDigitizerMultiRecordAcquisition	Extension: IviDigitizer with the ability to do multi-record acquisitions.	Yes
IviDigitizerBoardTemperature	Extension: IviDigitizer with the ability to report the temperature of the digitizer.	No
IviDigitizerChannelFilter	Extension: IviDigitizer with the ability to control the channel input filter frequency.	Yes

Table 1 Supported IviDigitizer group

Group name	Description	Supported
IviDigitizerChannelTemperature	Extension: IviDigitizer with the ability to report the temperature of individual digitizer channels.	No
IviDigitizerTimeInterleavedChannels	Extension: IviDigitizer with the ability to combine two or more input channels to achieve higher acquisitions rates and/or record lengths.	No
IviDigitizerDataInterleavedChannels	Extension: IVIdigitizer with the ability to interleave the data from two or more input channels, usually to create complex (I/Q) data.	No
IviDigitizerReferenceOscillator	Extension: IviDigitizer with the ability to use an external reference oscillator.	No
IviDigitizerSampleClock	Extension: IviDigitizer with the ability to use an external sample clock.	No
IviDigitizerSampleMode	Extension: IviDigitizer with the ability to control whether the digitizer is using real-time or equivalent-time sampling	No
IviDigitizerSelfCalibration	Extension: IviDigitizer with the ability to perform self-calibration.	No
IviDigitizerDownconversion	Extension: IviDigitizer with the ability to do frequency translation or downconversion in hardware.	No
IviDigitizerArm	Extension: IviDigitizer with the ability to arm on positive or negative edges.	No
IviDigitizerMultiArm	Extension: IviDigitizer with the ability to arm on one or more sources.	No
IviDigitizerGlitchArm	Extension: IviDigitizer with the ability to arm on glitches.	No
IviDigitizerRuntArm	Extension: IviDigitizer with the ability to arm on runts.	No
IviDigitizerSoftwareArm	Extension: IviDigitizer with the ability to arm acquisitions.	No
IviDigitizerTVArm	Extension: IviDigitizer with the ability to arm on standard TV signals.	No
IviDigitizerWidthArm	Extension: IviDigitizer with the ability to arm on a variety of conditions regarding pulse widths.	No
IviDigitizerWindowArm	Extension: IviDigitizer with the ability to arm on signals entering or leaving a defined voltage range.	No
IviDigitizerTriggerModifier	Extension: IviDigitizer with the ability to perform an alternative triggering function in the event that the specified trigger event does not occur.	No
IviDigitizerMultiTrigger	Extension: IviDigitizer with the ability to trigger on one or more sources.	Yes
IviDigitizerPretriggerSamples	Extension: IviDigitizer with the ability to specify a number of samples to fill up the data buffer with pre-trigger data.	Yes
IviDigitizerTriggerHoldoff	Extension: IviDigitizer with the ability to specify a length of time after the digitizer detects a trigger during which the digitizer ignores additional triggers.	Yes
IviDigitizerGlitchTrigger	Extension: IviDigitizer with the ability to trigger on glitches.	No
IviDigitizerRuntTrigger	Extension: IviDigitizer with the ability to trigger on runts.	No
IviDigitizerSoftwareTrigger	Extension: IviDigitizer with the ability to trigger acquisitions.	Yes

Table 1 Supported IviDigitizer group

Group name	Description	Supported
IviDigitizerTVTrigger	Extension: IviDigitizer with the ability to trigger on standard television signals.	No
IviDigitizerWidthTrigger	Extension: IviDigitizer with the ability to trigger on a variety of conditions regarding pulse widths.	No
IviDigitizerWindowTrigger	Extension: IviDigitizer with the ability to trigger on signals entering or leaving a defined voltage range.	Yes

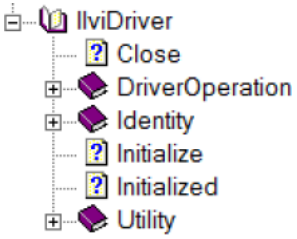
– **Instrument-Specific Hierarchy**

  – The M9217A PXIe 2-CH Digitizer Instrument-Specific Hierarchy has IKtM9217 at the root (where KtM9217 is the driver name).

    IKtM9217 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 IIviDriver interfaces are incorporated into both hierarchies: Class-Compliant Hierarchy and Instrument-Specific Hierarchy.

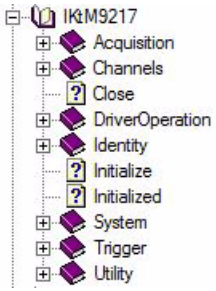
  The IIviDriver is the root interface for IVI Inherent Capabilities which are what the IVI Foundation has established as a set of functions and attributes that all IVI Drivers must include – irrespective of which IVI instrument class the driver supports. These common functions and attributes are called IVI inherent capabilities and they are documented in IVI-3.2 – Inherent Capabilities Specification.



IIviDriver
Close
DriverOperation
Identity
Initialize
Initialized
Utility

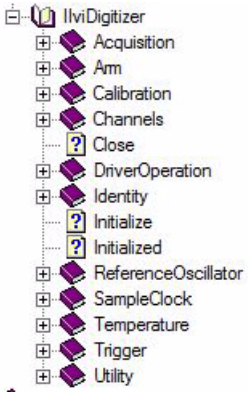
Instrument-Specific Hierarchies for the M9217A PXIe 2-CH Digitizer

The following table lists the Instrument-Specific Hierarchy interfaces for the M9217A PXIe 2-CH Digitizer.

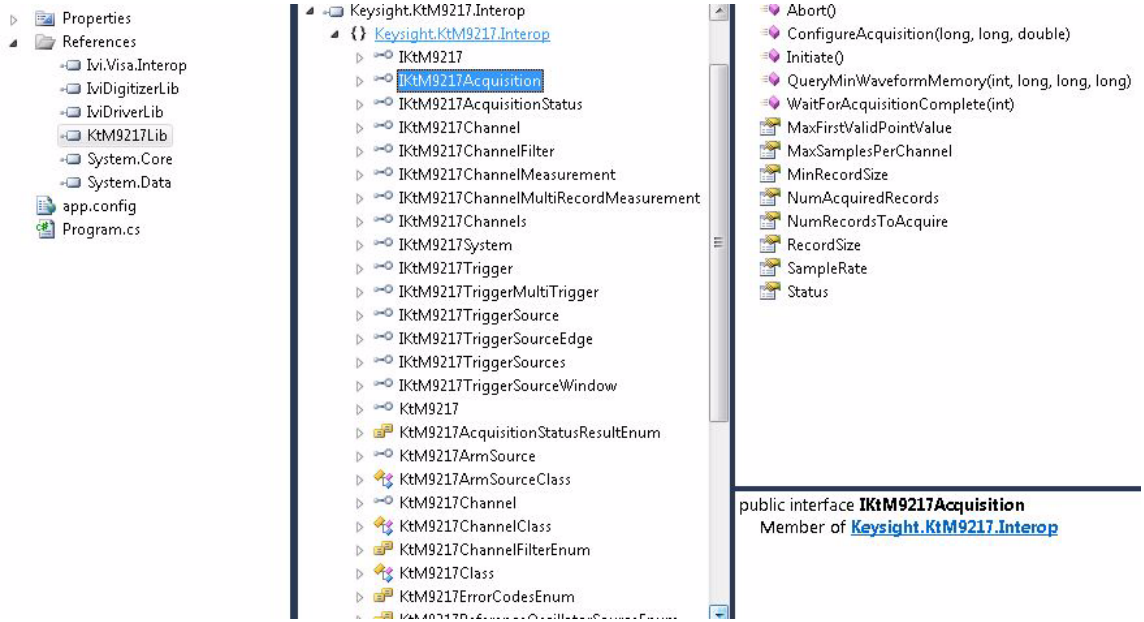
Keysight M9217A PXIe 2-CH Digitizer **Instrument-Specific Hierarchy**
KtM9217 is the driver name
IKtM9217 is the root interface


Instrument-Class Hierarchies for the M9217A PXIe 2-CH Digitizer

The following table lists the instrument-class hierarchy interfaces for the M9217A PXIe 2-CH Digitizer.

Keysight M9217A PXIe 2-CH Digitizer **Instrument-Class Hierarchy**
IviDigitizer Class
IIviDigitizer is the root interface


NOTE To view interfaces available in the M9217A PXIe 2-CH Digitizer, right-click the KtM9217Lib library file, in the References folder, from the Solution Explorer window and select View in Object Browser.



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”
  - I<ClassName>

- Examples: 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 IVI-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

Tutorial: Create a Project with IVI-COM Using C#

This tutorial will walk through the various steps required to create a console application using Visual Studio and C#. It demonstrates how to instantiate two driver instances, set the resource names and various initialization values, initialize the two driver instances, print various driver properties to a console for each driver instance, check drivers for errors and report the errors if any occur, and close both drivers.

- **Step 1.** Create a "Console Application"
- **Step 2.** Add References
- **Step 3.** Add using Statements
- **Step 4.** Create an Instance
- **Step 5.** Initialize the Instance
- **Step 6.** Write the Program Steps (Create a Square Waveform Output)
- **Step 7.** Close the Instance

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# **Console Application**.

Enter “M9217Properties” 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.

2 Select **Project** and click **Add Reference**. The Add Reference dialog appears.

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.

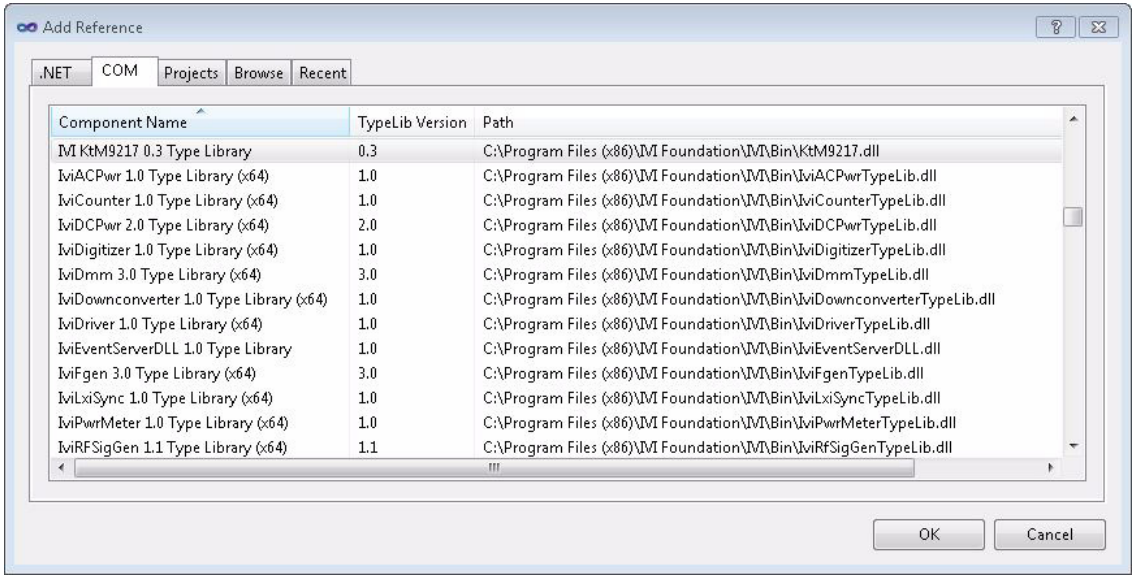
Step 2 – Add references

In order to access the M9217A PXIe 2-CH Digitizer driver interfaces, a reference to its driver (DLL) must be created.

1 In **Solution Explorer**, right-click on **References** and select **Add Reference**.

2 From the **Add Reference** dialog, select the **COM** tab.

3 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 M9217A PXIe 2-CH Digitizer (as listed in the previous section titled "**Before Programming, Install Hardware, Software, and Licenses**"), its IVI Driver 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.
- 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**".

4 Scroll to the IVI section and, using Shift-Ctrl, select the following type libraries then select OK.

IVI KtM9217 1.0 Type Library

NOTE When any of the references for the M9217A PXIe 2-CH Digitizer are added, the `IVIDriver 1.0 Type Libary` and `IviDigitizer 1.0 Type Library` is also automatically added. This is visible as `IviDriverLib` and `IviDigitizerTypeLib` under the project Reference; this reference houses the interface definitions for IVI inherent capabilities which are located in the file IviDriverTypeLib.dll and IviDigitizerTypeLib.dll (dynamically linked library).

5 These selected type libraries appear under the References node, in Solution Explorer, as:

NOTE Your program looks the same as it did before you added the References, but the difference is that the IVI Drivers that you added References to 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; the following example illustrates how to add `using` statements.

To access the IVI Drivers without having to specify or type the full path of each interface or enum

These using statements should be added to your program:

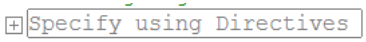
```
using Ivi.Driver.Interop;
using Keysight.KtM9217.Interop;
```

NOTE You can create sections of code in your program that can be expanded and collapsed by surrounding the code with `#region` and `#endregion` keywords. Selecting the – and + symbols allows the region to be collapsed and expanded.

Collapsed

```
#region Specify using Directives
    using System;
    using System.Collections.Generic;
    using System.Diagnostics;
    using System.Text;
    using Keysight.KtM9217.Interop;
#endregion
```

Expanded



Step 4 – Create instances of the IVI-COM drivers

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

- Direct Instantiation
- COM Session Factory

Since the M9217A PXIe 2-CH Digitizer is considered a NoClass module (because it does not belong to one of the 13 IVI Classes), the COM Session Factory is not used to create instances of its IVI-COM drivers. So, the M9217A PXIe 2-CH Digitizer IVI-COM driver uses direct instantiation.

To create driver instances

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

```
IKtM9217 driver = new KtM9217();
```

Step 5 – Initialize the driver instances

`Initialize()` 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 (see Initialize Options below). In this example, we prepare the `Initialize()` method by defining only a few of the parameters, then we call the `Initialize()` method with those parameters:

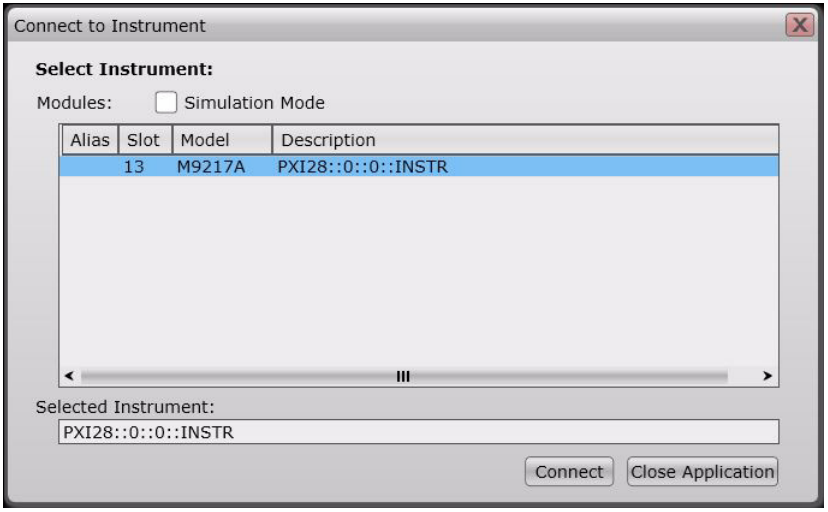
To determine the ResourceName

- If you are using Simulate Mode, you can set the ResourceName 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 Keysight Connection Expert, National Instruments Measurement and Automation Explorer (MAX), or you can use the Keysight product’s SFP to get the physical Resource Name string.

  Using the M9217A SFP, you might get the following Resource Name address string.



Module name	M9217A PXIe 2-CH Digitizer
Slot number	13
VISA address	PXI28::0::0::INSTR

```
string resourceDesc = "PXI28::0::0::INSTR";
```

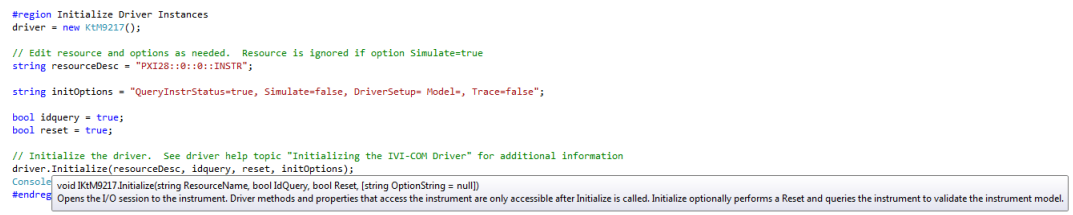
Set the Initialize() Parameters

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.

```
string resourceDesc = "PXI28::0::0::INSTR";
string initOptions = "QueryInstrStatus=true, Simulate=false,
DriverSetup= Model=, Trace=false";
bool idquery = true;
bool reset = true;
```

Call the Initialize() method with the set parameters

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



The above example 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.

Understanding Initialize options

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

Property type and example value	Description of property
`string ResourceName = PXI[bus]::device[::function][::INSTR]` `string ResourceName = "PXI28::0::0::INSTR";`	`ResourceName` The driver is typically initialized using a physical resource name descriptor, often a VISA resource descriptor. See the above procedure: “**To determine the ResourceName**”
`bool IdQuery = true;`	`IdQuery` Setting the **ID query** to false prevents the driver from verifying that the connected instrument is the one the driver was written for because if `IdQuery` is set to true, this will query the instrument model and fail initialization if the model is not supported by the driver.
`bool Reset = true;`	`Reset` Setting **Reset** to true tells the driver to initially reset the instrument.

Property type and example value	Description of property
`string OptionString = "QueryInstrStatus=true, Simulate=true,`	`OptionString` Setup the following initialization options: - `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 tells the driver that it should 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.)
`DriverSetup= Trace=false";`	- `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=M9217A` (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:

KtM9217A IVI Driver Reference

Start > **All Programs** > **Keysight Instrument Drivers** > **IVI-COM-C Drivers** > **KtM9217 Dynamic DAC**

Step 6 – Write the program steps

At this point, you can add program steps that use the driver instances to perform tasks.

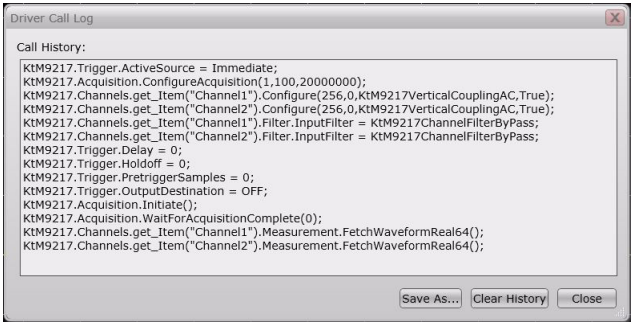
Example: Using the SFP to write program commands

You may find it useful when developing a program to use the instrument's SFP "Driver Call Log"; this driver call log is used to view a list of driver calls that have been performed when changes are made to the controls on the SFP.

In this example, open the SFP for the M9217A PXIe 2-CH Digitizer and perform the following steps:

1. Configure trigger source settings
2. Configure acquisition settings
3. Configure channel settings
4. Apply initiate
5. Wait for acquisition to complete
6. Fetch acquired data

Property type and example value	Description of property
KtM9217 is the driver name used by the SFP.	`driver` is the instance of the driver that is used in this example. This instance would have been created in, "**Step 4 – Create instances of the IVI-COM drivers**". `IKtM9217 driver = new KtM9217();`

	`//Set hardware trigger source to immediate mode`
	`KtM9217.Trigger.ActiveSource = Immediate;`
	
	`//Set 1 record with 100 record size and 20M Sa/s to acquire`
	`KtM9217.Acquisition.ConfigureAcquisition (1,100,20000000);`
	
	`//Set 256 Voltage Range, AC coupling for channel 1 and channel 2`
	`KtM9217.Channels.get_Item("Channel1").Configure(256,0,KtM9217VerticalCouplingAC, True);KtM9217.Channels.get_Item("Channel 2").Configure(256,0,KtM9217VerticalCouplingAC,True);`
	
	`//Set channel 1 and channel 2 to bypass filter`
	`KtM9217.Channels.get_Item("Channel1").Filter.InputFilter = KtM9217ChannelFilterByPass;`
	`KtM9217.Channels.get_Item("Channel2").Filter.InputFilter = KtM9217ChannelFilterByPass;`
	
	`//Configure trigger setting`
	`KtM9217.Trigger.Delay = 0;`
	`KtM9217.Trigger.Holdoff = 0;`
	`KtM9217.Trigger.PretriggerSamples = 0;`
	
	`//Disable OutputDestination`
	`KtM9217.Trigger.OutputDestination = OFF;`
	
	`//Apply initiate`
	`KtM9217.Acquisition.Initiate();`
	
	`//Wait for acquisition to complete with infinite timeout setting`
	`KtM9217.Acquisition.WaitForAcquisitionComplete(0);`
	
	`//Fetch Channel 1 and channel 2 data`
	`KtM9217.Channels.get_Item("Channel1").Measurement.FetchWaveformReal64();`
	`KtM9217.Channels.get_Item("Channel2").Measurement.FetchWaveformReal64();`

Step 7 – Close the driver

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

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.

```
{
     if (driver != null && driver.Initialized)
     {
        #region Close Driver Instances
        driver.Close();
        Console.WriteLine("Driver Closed");
        #endregion
     }
}
```

Step 8 – Building and running a complete example program using Visual C#

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. But, 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**.

   Alternate: From the Debug menu, click Start Debugging or press the F5 key.

Example programs may be found by selecting:

C:\Program Files\IVI Foundation\IVI\Drivers\KtM9217\Examples

or

C:\Program Files (x86)\IVI Foundation\IVI\Drivers\KtM9217\Examples

Example program 1: How to print the driver properties and close the driver sessions

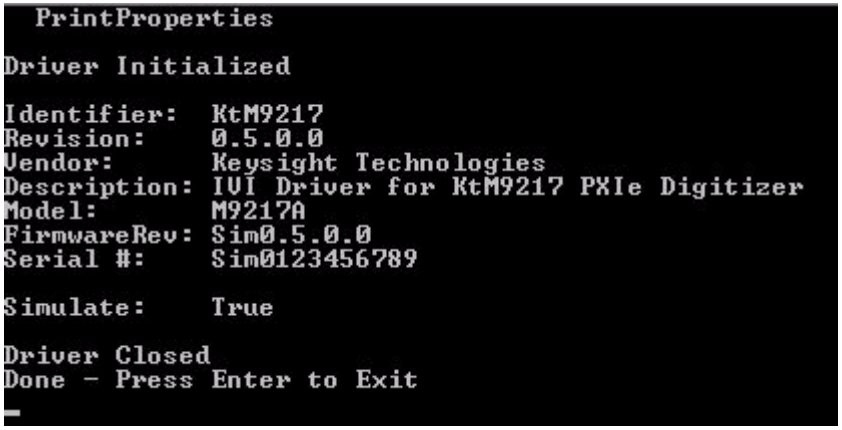
The following example code builds on the previously presented "**Tutorial: Create a Project with IVI-COM Using C#**" and demonstrates how to instantiate driver instances, set the resource names and various initialization values, initialize the two driver instances, print various driver properties for each driver instance, check drivers for errors and report the errors if any occur, and close the drivers.

```
// Copy the following example code and compile it as a C# Console Application
// Example__KtM9217_PrintProperties.cs
Specify using Directives

namespace KtM9217_PrintProperties
{
    class Program
    {
        [STAThread]
        public static void Main(string[] args)
        {
            Console.WriteLine("  PrintProperties");
            Console.WriteLine();
            KtM9217 driver = null;

            try
            {
                Initialize Driver Instances

                Print Driver Properties

                Perform Tasks

            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
            finally
            {
                if (driver != null && driver.Initialized)
                {
                    Close Driver Instances
                }
            }

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

Example program 1: How to print the driver properties and close the driver sessions

```
// Copy the following example code and compile it as a C# Console Application
// Example__KtM9217_PrintProperties.cs
#region Specify using Directives
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Text;
using Keysight.KtM9217.Interop;
#endregion

namespace KtM9217_PrintProperties
{
    class Program
    {
        [STAThread]
        public static void Main(string[] args)
        {
            Console.WriteLine("  PrintProperties");
            Console.WriteLine();
            KtM9217 driver = null;

            try
            {
                #region Initialize Driver Instances
                driver = new KtM9217();

                // Edit resource and options as needed.  Resource is ignored if
option Simulate=true
                string resourceDesc = "PXI28::0::0::INSTR";

                string initOptions = "QueryInstrStatus=true, Simulate=false,
DriverSetup= Model=, Trace=false";

                bool idquery = true;
                bool reset = true;

                // Initialize the driver.  See driver help topic "Initializing
the IVI-COM Driver" for additional information
                driver.Initialize(resourceDesc, idquery, reset, initOptions);
                Console.WriteLine("Driver Initialized\n");
                #endregion
```

```
                #region Print Driver 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);
                #endregion

                #region Perform Tasks
                // TO DO: Exercise driver methods and properties.
                // Put your code here to perform tasks with module.
                #endregion

            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
            finally
            {
                if (driver != null && driver.Initialized)
                {
                    #region Close Driver Instances
                    driver.Close();
                    Console.WriteLine("Driver Closed");
                    #endregion
                }
            }

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



NOTE

Disclaimer

You have a royalty-free right to use, modify, reproduce and distribute this Sample Application (and/or any modified version) in any way you find useful, provided that you agree that Keysight Technologies has no warranty, obligations or liability for any Sample Application Files.

Keysight Technologies provides programming examples for illustration only. This sample program assumes that you are familiar with the programming language being demonstrated and the tools used to create and debug procedures.

Keysight Technologies support engineers can help explain the functionality of Keysight Technologies software components and associated commands, but they will not modify these samples to provide added functionality or construct procedures to meet your specific needs.

Example program 2: How to perform waveform acquisition and returning waveform, using the immediate trigger

The following example code demonstrates how to instantiate a driver instance, set the resource name and various initialization values, as well as initialize the driver instances:

1. Apply changes to hardware.
2. Apply initiate acquisition.
3. Wait for acquisition to complete.
4. Fetch waveform data.
5. Report errors if any occur, and close the drivers.

```
// Copy the following example code and compile it as a C# Console Application
// Example__KtM9217_Immediate.cs
Specify using Directives

namespace KtM9217_Immediate
{
    class Program
    {
        [STAThread]
        public static void Main(string[] args)
        {
            Console.WriteLine("  Immediate");
            Console.WriteLine();
            KtM9217 driver = null;

            try
            {
                Initialize Driver Instances

                ActiveSource Settings

                Acquisition Settings

                Channels Settings - Channel 1

                Channels Settings - Channel 2

                Apply waveform source

                Initiates waveform acquisition

                Wait for acquisition to complete

                Fetch waveform data

            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
            finally
            {
                if (driver != null && driver.Initialized)
                {
                    Close Driver Instances
                }
            }

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

Pseudo-code of how to perform waveform acquisition and returning waveform, using the immediate trigger

1. Configure hardware `ActiveSource` to `"Immediate"`.
2. Configure `Acquisition` settings:
   - `1 record`
   - `16000 record size`
   - `20 MSa/s`
3. Configure `Channel 1` settings:
   - `16 V voltage range`
   - `AC coupling`
4. Configure `Channel 2` settings:
   - `32 V voltage range`
   - `DC coupling`
5. Initiate acquisition.
6. Wait for acquisition to complete.
7. Fetch waveform data.

Example program 2: How to perform waveform acquisition and returning waveform, using the immediate trigger

```
// Copy the following example code and compile it as a C# Console Application
// Example__KtM9217_Immediate.cs
#region Specify using Directives
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Text;
using Keysight.KtM9217.Interop;
#endregion

namespace KtM9217_Immediate
{
    class Program
    {
        [STAThread]
        public static void Main(string[] args)
        {
            Console.WriteLine("  Immediate");
            Console.WriteLine();
```

```
            KtM9217 driver = null;

            try
            {
                #region Initialize Driver Instances
                driver = new KtM9217();

                // Edit resource and options as needed.  Resource is ignored if
option Simulate=true
                string resourceDesc = "PXI28::0::0::INSTR";

                string initOptions = "QueryInstrStatus=true, Simulate=false,
DriverSetup= Model=, Trace=false";

                bool idquery = true;
                bool reset = true;

                // Initialize the driver.  See driver help topic "Initializing
the IVI-COM Driver" for additional information
                driver.Initialize(resourceDesc, idquery, reset, initOptions);
                Console.WriteLine("Driver Initialized\n");
                #endregion

                #region ActiveSource Settings
                Console.WriteLine("Configuring ActiveSource source to
Immediate\n");
                driver.Trigger.ActiveSource = ("Immediate");
                #endregion

                #region Acquisition Settings
                Console.WriteLine("Configuring Acquisition...");

                Console.WriteLine("Number of records to acquire:    1");
                driver.Acquisition.NumRecordsToAcquire = 1;

                Console.WriteLine("Number of record size:           16000");
                driver.Acquisition.RecordSize = 16000;

               Console.WriteLine("Rate of the sample clock:       20000000\n");
                driver.Acquisition.SampleRate = 20000000;
                #endregion

                #region Channels Settings - Channel 1
                Console.WriteLine("Configuring Channel 1...");
```

```
                Console.WriteLine("Voltage range:   16V");
                driver.Channels.get_Item("Channel1").Range = 16;

                Console.WriteLine("Coupling:        AC\n");
                driver.Channels.get_Item("Channel1").Coupling =
KtM9217VerticalCouplingEnum.KtM9217VerticalCouplingAC;
                #endregion

                #region Channels Settings - Channel 2
                Console.WriteLine("Configuring Channel 2...");
                Console.WriteLine("Voltage range:   32V");
                driver.Channels.get_Item("Channel2").Range = 32;

                Console.WriteLine("Coupling:        DC\n");
                driver.Channels.get_Item("Channel2").Coupling =
KtM9217VerticalCouplingEnum.KtM9217VerticalCouplingDC;
                #endregion

                #region Apply waveform source
                /// Apply same waveform source to Digitizer Channel1 and Channel2
                /// Waveform Generator Settings
                /// Amplitude:  10Vpp
                /// Offset:     5V
                /// Freqeuncy:  5kHz
                /// Waveform:   Sinusoidal
                Console.WriteLine("Apply waveform source...\n");
                #endregion

                #region Initiates waveform acquisition
                Console.WriteLine("The hardware leaves the Idle state and waits
for trigger to acquire waveform\n");
                driver.Acquisition.Initiate();
                #endregion

                #region Wait for acquisition to complete
                Console.WriteLine("Wait for acquisition to complete -
MaxTimeMilliseconds (Infinite)\n");
                driver.Acquisition.WaitForAcquisitionComplete(0);
                #endregion

                #region Fetch waveform data
                double[] WaveformArray_Ch1 = {};
                long ActualPoints_Ch1 = 0;
                long FirstValidPoint_Ch1 = 0;
```

```
                double InitialXOffset_Ch1 = 0;
                double InitialXTimeSeconds_Ch1 = 0.0;
                double InitialXTimeFraction_Ch1 = 0.0;
                double XIncrement_Ch1 = 0.0;

                double[] WaveformArray_Ch2 = { };
                long ActualPoints_Ch2 = 0;
                long FirstValidPoint_Ch2 = 0;
                double InitialXOffset_Ch2 = 0;
                double InitialXTimeSeconds_Ch2 = 0.0;
                double InitialXTimeFraction_Ch2 = 0.0;
                double XIncrement_Ch2 = 0.0;
                Console.WriteLine("Fetch Channel 1 acquired waveform...");

driver.Channels.get_Item("Channel1").Measurement.FetchWaveformReal64(ref
WaveformArray_Ch1, ref ActualPoints_Ch1, ref FirstValidPoint_Ch1, ref
InitialXOffset_Ch1, ref InitialXTimeSeconds_Ch1, ref InitialXTimeFraction_Ch1,
ref XIncrement_Ch1);


                Console.WriteLine("Fetch Channel 2 acquired waveform...");

driver.Channels.get_Item("Channel2").Measurement.FetchWaveformReal64(ref
WaveformArray_Ch2, ref ActualPoints_Ch2, ref FirstValidPoint_Ch2, ref
InitialXOffset_Ch2, ref InitialXTimeSeconds_Ch2, ref InitialXTimeFraction_Ch2,
ref XIncrement_Ch2);
                #endregion


            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
            finally
            {
                if (driver != null && driver.Initialized)
                {
                    #region Close Driver Instances
                    driver.Close();
                    Console.WriteLine("Driver Closed");
                    #endregion
                }
            }

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

```
        }
    }
}
```

```
  Immediate

Driver Initialized

Configuring ActiveSource source to Immediate

Configuring Acquisition...
Number of records to acquire:    1
Number of record size:           16000
Rate of the sample clock:        20000000

Configuring Channel 1...
Voltage range:   16V
Coupling:        AC

Configuring Channel 2...
Voltage range:   32V
Coupling:        DC

Apply waveform source...

The hardware leaves the Idle state and waits for trigger to acquire waveform

Wait for acquisition to complete - MaxTimeMilliseconds (Infinite)

Fetch Channel 1 acquired waveform...
Fetch Channel 2 acquired waveform...
Driver Closed
Done - Press Enter to Exit
```

Figure 1-1 SFP waveform results with example program 2 settings

Example program 3: How to perform waveform acquisition and returning waveform, using the channel trigger

The following example code demonstrates how to instantiate a driver instance, set the resource name and various initialization values, as well as initialize the driver instances:

1. Apply changes to hardware.
2. Apply initiate acquisition.
3. Wait for trigger and wait for acquisition to complete.
4. Fetch waveform data.
5. Report errors if any occur, and close the drivers.

```
// Copy the following example code and compile it as a C# Console Application
// Example__KtM9217_Channel_Trigger.cs
Specify using Directives

namespace KtM9217_Channel_Trigger
{
    class Program
    {
        [STAThread]
        public static void Main(string[] args)
        {
            Console.WriteLine("  Channel Trigger");
            Console.WriteLine();
            KtM9217 driver = null;

            try
            {
                Initialize Driver Instances

                ActiveSource Settings

                Acquisition Settings

                Channels Settings - Channel 1

                Channels Settings - Channel 2

                Configure waveform source

                Trigger Settings

                Initiates waveform acquisition

                Wait for acquisition to complete

                Output waveform source

                Fetch waveform data

            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
            finally
            {
                if (driver != null && driver.Initialized)
                {
                    Close Driver Instances
                }
            }

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

Pseudo-code of how to perform waveform acquisition and returning waveform, using the channel trigger

1. Configure hardware `ActiveSource` to `"Channel1"`.
2. Configure `Acquisition` settings:
   - `1 record`
   - `10000 record size`
   - `1 MSa/s`
3. Configure `Channel 1` settings:
   - `16 V voltage range`
   - `AC coupling`
4. Configure `Channel 2` settings:
   - `64 V voltage range`
   - `DC coupling`
5. Configure `Trigger` settings:
   - `1000 pre trigger samples`
   - `Trigger Type: Edge`
   - `Trigger Level: 3 V`
   - `Trigger Slope: Positive`
   - `Trigger Hysteresis: 0.4 V`
6. Initiate acquisition.
7. Wait for trigger and wait for acquisition to complete.
8. Fetch waveform data.

Example program 3: How to perform waveform acquisition and returning waveform, using the channel trigger

```
// Copy the following example code and compile it as a C# Console Application
// Example__KtM9217_Channel_Trigger.cs
#region Specify using Directives
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Text;
using Keysight.KtM9217.Interop;
#endregion

namespace KtM9217_Channel_Trigger
{
```

```
    class Program
    {
        [STAThread]
        public static void Main(string[] args)
        {
            Console.WriteLine("  Channel Trigger");
            Console.WriteLine();
            KtM9217 driver = null;

            try
            {
                #region Initialize Driver Instances
                driver = new KtM9217();

                // Edit resource and options as needed.  Resource is ignored if
option Simulate=true
                string resourceDesc = "PXI28::0::0::INSTR";

                string initOptions = "QueryInstrStatus=true, Simulate=false,
DriverSetup= Model=, Trace=false";

                bool idquery = true;
                bool reset = true;

                // Initialize the driver.  See driver help topic "Initializing
the IVI-COM Driver" for additional information
                driver.Initialize(resourceDesc, idquery, reset, initOptions);
                Console.WriteLine("Driver Initialized\n");
                #endregion

                #region ActiveSource Settings
                Console.WriteLine("Configuring ActiveSource source to
Channel1\n");
                driver.Trigger.ActiveSource = ("Channel1");
                #endregion

                #region Acquisition Settings
                Console.WriteLine("Configuring Acquisition...");

                Console.WriteLine("Number of records to acquire:    1");
                driver.Acquisition.NumRecordsToAcquire = 1;

                Console.WriteLine("Number of record size:           10000");
                driver.Acquisition.RecordSize = 10000;
```

```
                Console.WriteLine("Rate of the sample clock:        1000000\n");
                driver.Acquisition.SampleRate = 1000000;
                #endregion

                #region Channels Settings - Channel 1
                Console.WriteLine("Configuring Channel 1...");
                Console.WriteLine("Voltage range:   16V");
                driver.Channels.get_Item("Channel1").Range = 16;

                Console.WriteLine("Coupling:        AC\n");
                driver.Channels.get_Item("Channel1").Coupling =
KtM9217VerticalCouplingEnum.KtM9217VerticalCouplingAC;
                #endregion

                #region Channels Settings - Channel 2
                Console.WriteLine("Configuring Channel 2...");
                Console.WriteLine("Voltage range:   64V");
                driver.Channels.get_Item("Channel2").Range = 64;

                Console.WriteLine("Coupling:        DC\n");
                driver.Channels.get_Item("Channel2").Coupling =
KtM9217VerticalCouplingEnum.KtM9217VerticalCouplingDC;
                #endregion

                #region Configure waveform source
                /// Apply waveform sources to Digitizer Channel1 and Channel2
                /// Waveform Generator Settings - Channel1
                /// Amplitude:  10Vpp
                /// Offset:     0V
                /// Freqeuncy:  5kHz
                /// Waveform:   Pulse Wave
                ///
                /// Waveform Generator Settings - Channel2
                /// Amplitude:  0 - 30V
                /// Offset:     0V
                /// Freqeuncy:  5kHz
                /// Waveform:   Arbitrary Waveform
                Console.WriteLine("Configure waveform source...\n");
                #endregion

                #region Trigger Settings
                Console.WriteLine("Configuring Trigger Settings (Channel1)...");
```

```
                Console.WriteLine("PretriggerSamples:         1000");
                driver.Trigger.PretriggerSamples = 1000;
                Console.WriteLine("Trigger Type:              Edge");
                driver.Trigger.Sources.get_Item("Channel1").Type =
KtM9217TriggerTypeEnum.KtM9217TriggerEdge;
                Console.WriteLine("Trigger Edge Level:        3V");
                driver.Trigger.Sources.get_Item("Channel1").Level = 3;
                Console.WriteLine("Trigger Edge Hysteresis:   0.4V");
                driver.Trigger.Sources.get_Item("Channel1").Hysteresis = 0.4;
                Console.WriteLine("Trigger Edge Slope:        Positive\n");
                driver.Trigger.Sources.get_Item("Channel1").Edge.Slope =
KtM9217TriggerSlopeEnum.KtM9217TriggerSlopePositive;
                #endregion

                #region Initiates waveform acquisition
                Console.WriteLine("The hardware leaves the Idle state and waits
for trigger to acquire waveform\n");
                driver.Acquisition.Initiate();
                #endregion

                #region Wait for acquisition to complete
                Console.WriteLine("Wait for acquisition to complete -
MaxTimeMilliseconds (2000ms)\n");
                driver.Acquisition.WaitForAcquisitionComplete(2000);
                #endregion

                #region Output waveform source
                // Output waveform sources to Digitizer Channel1 and Channel2
                Console.WriteLine("Output waveform source...\n");
                #endregion

                #region Fetch waveform data
                double[] WaveformArray_Ch1 = {};
                long ActualPoints_Ch1 = 0;
                long FirstValidPoint_Ch1 = 0;
                double InitialXOffset_Ch1 = 0;
                double InitialXTimeSeconds_Ch1 = 0.0;
                double InitialXTimeFraction_Ch1 = 0.0;
                double XIncrement_Ch1 = 0.0;

                double[] WaveformArray_Ch2 = { };
                long ActualPoints_Ch2 = 0;
                long FirstValidPoint_Ch2 = 0;
                double InitialXOffset_Ch2 = 0;
```

```
                double InitialXTimeSeconds_Ch2 = 0.0;
                double InitialXTimeFraction_Ch2 = 0.0;
                double XIncrement_Ch2 = 0.0;
                Console.WriteLine("Fetch Channel 1 acquired waveform...");

driver.Channels.get_Item("Channel1").Measurement.FetchWaveformReal64(ref
WaveformArray_Ch1, ref ActualPoints_Ch1, ref FirstValidPoint_Ch1, ref
InitialXOffset_Ch1, ref InitialXTimeSeconds_Ch1, ref InitialXTimeFraction_Ch1,
ref XIncrement_Ch1);


                Console.WriteLine("Fetch Channel 2 acquired waveform...");

driver.Channels.get_Item("Channel2").Measurement.FetchWaveformReal64(ref
WaveformArray_Ch2, ref ActualPoints_Ch2, ref FirstValidPoint_Ch2, ref
InitialXOffset_Ch2, ref InitialXTimeSeconds_Ch2, ref InitialXTimeFraction_Ch2,
ref XIncrement_Ch2);
                #endregion


            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
            finally
            {
                if (driver != null && driver.Initialized)
                {
                    #region Close Driver Instances
                    driver.Close();
                    Console.WriteLine("Driver Closed");
                    #endregion
                }
            }

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

```
  Channel Trigger

Driver Initialized

Configuring ActiveSource source to Channel1

Configuring Acquisition...
Number of records to acquire:    1
Number of record size:           10000
Rate of the sample clock:        1000000

Configuring Channel 1...
Voltage range:   16V
Coupling:        AC

Configuring Channel 2...
Voltage range:   64V
Coupling:        DC

Configure waveform source...

Configuring Trigger Settings (Channel1)...
PretriggerSamples:          1000
Trigger Type:               Edge
Trigger Edge Level:         3V
Trigger Edge Hysteresis:    0.4V
Trigger Edge Slope:         Positive

The hardware leaves the Idle state and waits for trigger to acquire waveform

Wait for acquisition to complete - MaxTimeMilliseconds (2000ms)

Output waveform source...

Fetch Channel 1 acquired waveform...
Fetch Channel 2 acquired waveform...
Driver Closed
Done - Press Enter to Exit
```

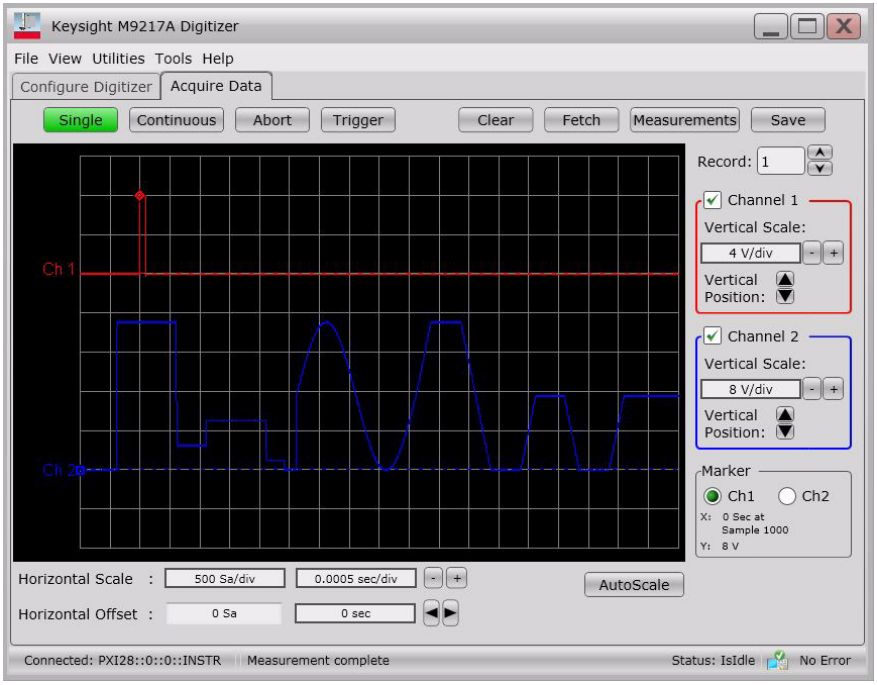


Figure 1-2 SFP waveform results with example program 3 settings

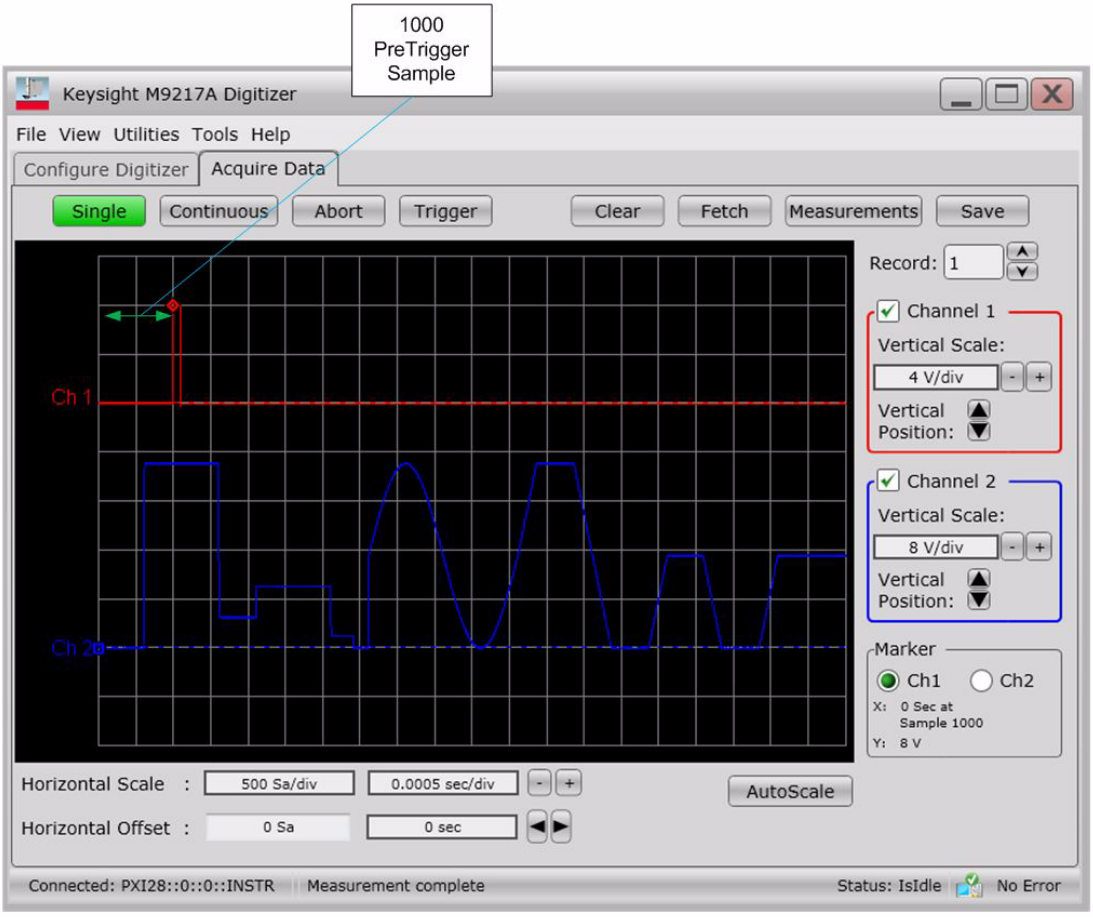


Figure 1-3 SFP waveform results with 1000 pre trigger sample indication

Example program 4: How to output a trigger pulse to EXT front connector when the trigger event signal is accepted at PXI_TRIG0.

The following example code demonstrates how to instantiate a driver instance, set the resource name and various initialization values, as well as initialize the driver instances:

1. Apply changes to hardware.
2. Apply initiate acquisition.
3. Wait for trigger and wait for acquisition to complete.
4. Fetch waveform data.
5. Report errors if any occur, and close the driver.

```
// Copy the following example code and compile it as a C# Console Application
// Example__KtM9217_OutputDestination.cs
Specify using Directives

namespace KtM9217_OutputDestination
{
    class Program
    {
        [STAThread]
        public static void Main(string[] args)
        {
            Console.WriteLine("  Channel Trigger");
            Console.WriteLine();
            KtM9217 driver = null;

            try
            {
                Initialize Driver Instances

                ActiveSource Settings

                Acquisition Settings

                Channels Settings - Channel 1

                Channels Settings - Channel 2

                Apply waveform source

                Trigger Settings

                Initiates waveform acquisition

                Wait for acquisition to complete

                Send a Trigger Pulse to PXI Bus0

                Verify Trigger Signal from EXT (OutputDestination)

                Fetch waveform data

            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
            finally
            {
                if (driver != null && driver.Initialized)
                {
                    #region Close Driver Instances
                    driver.Close();
                    Console.WriteLine("Driver Closed");
                    #endregion
                }
            }

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

Pseudo-code of how to output a trigger pulse to EXT front connector when the trigger event signal is accepted at PXI_TRIG0

1. Configure hardware `ActiveSource` to `"PXI_TRIG0"`.
2. Configure `Acquisition` settings:
   - `1 record`
   - `500 record size`
   - `100 kSa/s`
3. Configure `Channel 1` settings:
   - `32 V voltage range`
   - `AC coupling`
4. Configure `Channel 2` settings:
   - `16 V voltage range`
   - `AC coupling`
5. Configure `Trigger` settings:
   - `0 Pre Trigger Samples`
   - `Output Destination: External`
6. Initiate acquisition.
7. Wait for trigger and wait for acquisition to complete.
8. Fetch waveform data.

Example program 4: How to output a trigger pulse to EXT front connector when the trigger event signal is accepted at PXI_TRIG0

```
// Copy the following example code and compile it as a C# Console Application
// Example__KtM9217_OutputDestination.cs
#region Specify using Directives
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Text;
using Keysight.KtM9217.Interop;
#endregion

namespace KtM9217_OutputDestination
{
    class Program
    {
        [STAThread]
        public static void Main(string[] args)
        {
            Console.WriteLine("  Output Destination");
            Console.WriteLine();
            KtM9217 driver = null;

            try
            {
                #region Initialize Driver Instances
                driver = new KtM9217();

                // Edit resource and options as needed.  Resource is ignored if
option Simulate=true
                string resourceDesc = "PXI28::0::0::INSTR";

                string initOptions = "QueryInstrStatus=true, Simulate=false,
DriverSetup= Model=, Trace=false";

                bool idquery = true;
                bool reset = true;

                // Initialize the driver.  See driver help topic "Initializing
the IVI-COM Driver" for additional information
                driver.Initialize(resourceDesc, idquery, reset, initOptions);
                Console.WriteLine("Driver Initialized\n");
                #endregion
```

```
                #region ActiveSource Settings
                Console.WriteLine("Configuring ActiveSource source to
PXI_TRIG0\n");
                driver.Trigger.ActiveSource = ("PXI_TRIG0");
                #endregion

                #region Acquisition Settings
                Console.WriteLine("Configuring Acquisition...");

                Console.WriteLine("Number of records to acquire:    1");
                driver.Acquisition.NumRecordsToAcquire = 1;

                Console.WriteLine("Number of record size:           500");
                driver.Acquisition.RecordSize = 500;

                Console.WriteLine("Rate of the sample clock:        100000\n");
                driver.Acquisition.SampleRate = 100000;
                #endregion

                #region Channels Settings - Channel 1
                Console.WriteLine("Configuring Channel 1...");
                Console.WriteLine("Voltage range:   32V");
                driver.Channels.get_Item("Channel1").Range = 32;

                Console.WriteLine("Coupling:        AC\n");
                driver.Channels.get_Item("Channel1").Coupling =
KtM9217VerticalCouplingEnum.KtM9217VerticalCouplingAC;
                #endregion

                #region Channels Settings - Channel 2
                Console.WriteLine("Configuring Channel 2...");
                Console.WriteLine("Voltage range:   16V");
                driver.Channels.get_Item("Channel2").Range = 16;

                Console.WriteLine("Coupling:        AC\n");
                driver.Channels.get_Item("Channel2").Coupling =
KtM9217VerticalCouplingEnum.KtM9217VerticalCouplingAC;
                #endregion

                #region Apply waveform source
                /// Apply waveform sources to Digitizer Channel1 and Channel2
                /// Waveform Generator Settings - Channel1
```

```
                /// Amplitude:  20Vpp
                /// Offset:     0V
                /// Freqeuncy:  1kHz
                /// Waveform:   Square Wave
                /// DutyCycle:  80%
                ///
                /// Waveform Generator Settings - Channel2
                /// Amplitude:  10Vpp
                /// Offset:     0V
                /// Freqeuncy:  500Hz
                /// Waveform:   Square Wave
                /// DutyCycle:  80%
                Console.WriteLine("Apply waveform source...\n");
                #endregion

                #region Trigger Settings
                Console.WriteLine("Configuring Trigger Settings...");
                Console.WriteLine("PretriggerSamples:          0");
                driver.Trigger.PretriggerSamples = 0;

                Console.WriteLine("Output Destination:         External\n");
                driver.Trigger.OutputDestination = "External";
                #endregion

                #region Initiates waveform acquisition
                Console.WriteLine("The hardware leaves the Idle state and waits
for trigger to acquire waveform\n");
                driver.Acquisition.Initiate();
                #endregion

                #region Wait for acquisition to complete
                Console.WriteLine("Wait for acquisition to complete -
MaxTimeMilliseconds (36000ms)\n");
                driver.Acquisition.WaitForAcquisitionComplete(36000);
                #endregion

                #region Send a Trigger Pulse to PXI Bus0
                Console.WriteLine("Send a Trigger Pulse to PXI Bus0 ...\n");
                #endregion

                #region Verify Trigger Signal from EXT (OutputDestination)
                /*********************************************************
```

```
                When trigger event is accepted, a trigger pulse is outputted to
outputdestination
                ************************************************************/
                Console.WriteLine("Verify Trigger Signal from EXT
(OutputDestination)...\n");
                #endregion

                #region Fetch waveform data
                double[] WaveformArray_Ch1 = {};
                long ActualPoints_Ch1 = 0;
                long FirstValidPoint_Ch1 = 0;
                double InitialXOffset_Ch1 = 0;
                double InitialXTimeSeconds_Ch1 = 0.0;
                double InitialXTimeFraction_Ch1 = 0.0;
                double XIncrement_Ch1 = 0.0;

                double[] WaveformArray_Ch2 = { };
                long ActualPoints_Ch2 = 0;
                long FirstValidPoint_Ch2 = 0;
                double InitialXOffset_Ch2 = 0;
                double InitialXTimeSeconds_Ch2 = 0.0;
                double InitialXTimeFraction_Ch2 = 0.0;
                double XIncrement_Ch2 = 0.0;

                Console.WriteLine("Fetch Channel 1 acquired waveform...");

driver.Channels.get_Item("Channel1").Measurement.FetchWaveformReal64(ref
WaveformArray_Ch1, ref ActualPoints_Ch1, ref FirstValidPoint_Ch1, ref
InitialXOffset_Ch1, ref InitialXTimeSeconds_Ch1, ref InitialXTimeFraction_Ch1,
ref XIncrement_Ch1);

                Console.WriteLine("Fetch Channel 2 acquired waveform...");

driver.Channels.get_Item("Channel2").Measurement.FetchWaveformReal64(ref
WaveformArray_Ch2, ref ActualPoints_Ch2, ref FirstValidPoint_Ch2, ref
InitialXOffset_Ch2, ref InitialXTimeSeconds_Ch2, ref InitialXTimeFraction_Ch2,
ref XIncrement_Ch2);
                #endregion

            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
            finally
            {
```

```
                if (driver != null && driver.Initialized)
                {
                    #region Close Driver Instances
                    driver.Close();
                    Console.WriteLine("Driver Closed");
                    #endregion
                }
            }

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

```
  Output Destination

Driver Initialized

Configuring ActiveSource source to PXI_TRIG0

Configuring Acquisition...
Number of records to acquire:    1
Number of record size:           500
Rate of the sample clock:        100000

Configuring Channel 1...
Voltage range:   32V
Coupling:        AC

Configuring Channel 2...
Voltage range:   16V
Coupling:        AC

Apply waveform source...

Configuring Trigger Settings...
PretriggerSamples:          0
Output Destination:         External

The hardware leaves the Idle state and waits for trigger to acquire waveform

Wait for acquisition to complete - MaxTimeMilliseconds (36000ms)

Send a Trigger Pulse to PXI Bus0 ...

Verify Trigger Signal from EXT (OutputDestination)...

Fetch Channel 1 acquired waveform...
Fetch Channel 2 acquired waveform...
Driver Closed
Done - Press Enter to Exit
```

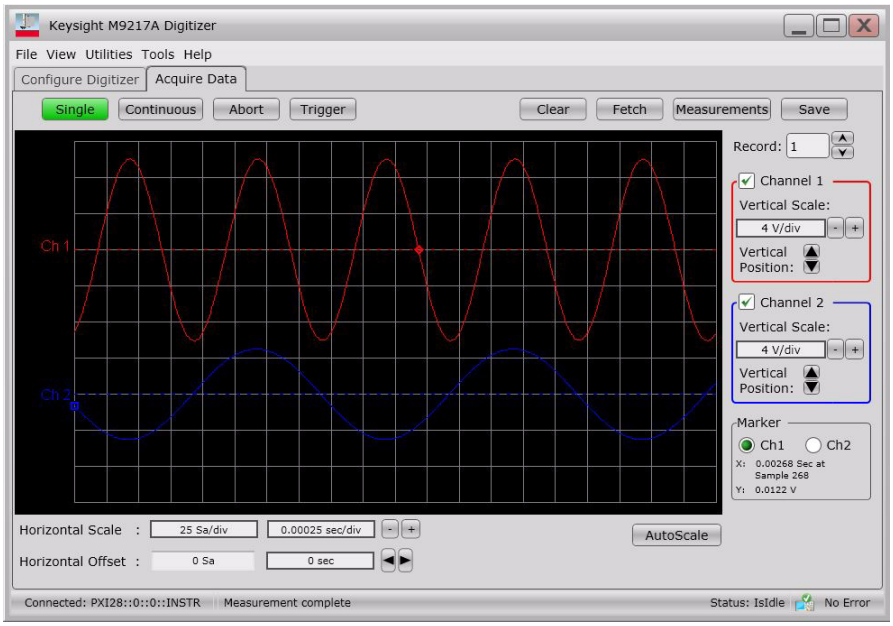


Figure 1-4 SFP waveform results with example program 4 settings

Figure 1-5 Oscilloscope signal pulse at KtM9217A PXIe 2-CH Digitizer external front connector when trigger event is accepted at PXI_TRIG0

The captured signal pulse is the trigger signal that appears at the KtM9217A PXIe 2-CH Digitizer external front connector only when the trigger event is accepted at PXI_TRIG0.

Example program 5: How to perform multiple waveform acquisition and returning waveforms, using the software trigger

The following example code demonstrates how to instantiate a driver instance, set the resource name and various initialization values, as well as initialize the driver instances:

1. Apply changes to hardware.
2. Apply initiate acquisition.
3. Apply SendSoftwareTrigger.
4. Fetch waveform data.
5. Report errors if any occur, and close the drivers.

```
// Copy the following example code and compile it as a C# Console Application
// Example__KtM9217_Multi_Record.cs
Specify using Directives

namespace KtM9217_Multi_Record
{
    class Program
    {
        [STAThread]
        public static void Main(string[] args)
        {
            Console.WriteLine("  Multi Record");
            Console.WriteLine();
            KtM9217 driver = null;

            try
            {
                Initialize Driver Instances

                ActiveSource Settings

                Acquisition Settings

                Channels Settings - Channel 1

                Configure waveform source

                Initiates waveform acquisition

                Send first software trigger

                Send second software trigger

                Send third software trigger

                Send fourth software trigger

                Query number of acquired records

                Fetch waveform data
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
            finally
            {
                if (driver != null && driver.Initialized)
                {
                    Close Driver Instances
                }
            }

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

Pseudo-code of how to perform multiple waveform acquisition and returning waveforms, using the software trigger

1. Configure hardware `ActiveSource` to `"Software"`.
2. Configure `Acquisition` settings:
   - `4 records`
   - `16000 record size`
   - `20 MSa/s`
3. Configure `Channel 1` settings:
   - `16 V voltage range`
   - `AC coupling`
4. Initiate acquisition.
5. Apply SendSoftwareTrigger four times.
6. Verify on number of acquired records.
7. Fetch waveform data.

Example program 5: How to perform multiple waveform acquisition and returning waveforms, using the software trigger

```
// Copy the following example code and compile it as a C# Console Application
// Example__KtM9217_Multi_Record.cs
#region Specify using Directives
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Text;
using Keysight.KtM9217.Interop;
using System.Threading;
#endregion

namespace KtM9217_Multi_Record
{
    class Program
    {
        [STAThread]
        public static void Main(string[] args)
        {
            Console.WriteLine("  Multi Record");
            Console.WriteLine();
            KtM9217 driver = null;

            try
            {
                #region Initialize Driver Instances
                driver = new KtM9217();

                // Edit resource and options as needed.  Resource is ignored if
option Simulate=true
                string resourceDesc = "PXI28::0::0::INSTR";

                string initOptions = "QueryInstrStatus=true, Simulate=false,
DriverSetup= Model=, Trace=false";

                bool idquery = true;
                bool reset = true;

                // Initialize the driver.  See driver help topic "Initializing
the IVI-COM Driver" for additional information
                driver.Initialize(resourceDesc, idquery, reset, initOptions);
                Console.WriteLine("Driver Initialized\n");
```

```
                #endregion

                #region ActiveSource Settings
                Console.WriteLine("Configuring ActiveSource source to
Software\n");
                driver.Trigger.ActiveSource = ("Software");
                #endregion

                #region Acquisition Settings
                Console.WriteLine("Configuring Acquisition...");

                Console.WriteLine("Number of records to acquire:    4");
                driver.Acquisition.NumRecordsToAcquire = 4;

                Console.WriteLine("Number of record size:          16000");
                driver.Acquisition.RecordSize = 16000;

               Console.WriteLine("Rate of the sample clock:       20000000\n");
                driver.Acquisition.SampleRate = 20000000;
                #endregion

                #region Channels Settings - Channel 1
                Console.WriteLine("Configuring Channel 1...");
                Console.WriteLine("Voltage range:   16V");
                driver.Channels.get_Item("Channel1").Range = 16;

                Console.WriteLine("Coupling:        AC\n");
                driver.Channels.get_Item("Channel1").Coupling =
KtM9217VerticalCouplingEnum.KtM9217VerticalCouplingAC;
                #endregion

                #region Configure waveform source
                /// Apply waveform sources to Digitizer Channel1
                /// Waveform Generator Settings - Channel1
                /// Amplitude:  10Vpp
                /// Offset:     0V
                /// Freqeuncy:  5kHz
                /// Waveform:   Sinusoidal
                Console.WriteLine("Apply waveform source...\n");
                #endregion

                #region Initiates waveform acquisition
```

```
                Console.WriteLine("The hardware leaves the Idle state and waits
for trigger to acquire waveform\n");
                driver.Acquisition.Initiate();
                #endregion

                #region Send first software trigger
                Console.WriteLine("Send first software trigger");
                driver.Trigger.SendSoftwareTrigger();
                Thread.Sleep(2); //Acquire data will take about 0.8ms and
additional buffer time
                #endregion

                #region Send second software trigger
                Console.WriteLine("Send second software trigger");
                driver.Trigger.SendSoftwareTrigger();
                Thread.Sleep(2); //Acquire data will take about 0.8ms and
additional buffer time
                #endregion

                #region Send third software trigger
                Console.WriteLine("Send third software trigger");
                driver.Trigger.SendSoftwareTrigger();
                Thread.Sleep(2); //Acquire data will take about 0.8ms and
additional buffer time
                #endregion

                #region Send fourth software trigger
                Console.WriteLine("Send fourth software trigger\n");
                driver.Trigger.SendSoftwareTrigger();
                Thread.Sleep(2); //Acquire data will take about 0.8ms and
additional buffer time
                #endregion

                #region Query number of acquired records
                Console.WriteLine("Number of Acquired records:    {0}\n",
driver.Acquisition.NumAcquiredRecords);
                #endregion

                #region Fetch waveform data

                if (driver.Acquisition.NumAcquiredRecords != 4)
                {
                    Console.WriteLine("Abort acquisition if number of acquired
record is not 4.\n");
                    driver.Acquisition.Abort();
```

```
                }
                else
                {

                    long FirstRecord = 0; //Fetch start with first record
                    long NumRecords = 4; //Fetch 4 record
                    long OffsetwithinRecord = 0; //0 point offset within record
                  long NumPointperRecord = 16000; //Fetch 16000 point per record
                    double[] WaveformArray = { };
                    long ActualRecords = 0;
                    long[] ActualPoints = { };
                    long[] FirstValidPoint = { };
                    double[] InitialXOffset = { };
                    double[] InitialXTimeSeconds = { };
                    double[] InitialXTimeFraction = { };
                    double XIncrement = 0.0;

                    //Fetch all 4 record 1 data
                    Console.WriteLine("Fetch Channel 1 all 4 acquired waveform
record...");

driver.Channels.get_Item("Channel1").MultiRecordMeasurement.FetchMultiRecordWav
eformReal64(FirstRecord, NumRecords, OffsetwithinRecord, NumPointperRecord, ref
WaveformArray, ref ActualRecords, ref ActualPoints, ref FirstValidPoint, ref
InitialXOffset, ref InitialXTimeSeconds, ref InitialXTimeFraction, ref
XIncrement);


                    //Fetch specific records with specificed number point per
record and offset withn record
                    FirstRecord = 1; //Fetch start with second record
                    NumRecords = 2; //Fetch 2 record
                  OffsetwithinRecord = 12000; //12000 point offset within record
                    NumPointperRecord = 4000; //Fetch 40000 point per record


                   Console.WriteLine("Fetch Channel 1 second and third acquired
waveform record...");

driver.Channels.get_Item("Channel1").MultiRecordMeasurement.FetchMultiRecordWav
eformReal64(FirstRecord, NumRecords, OffsetwithinRecord, NumPointperRecord, ref
WaveformArray, ref ActualRecords, ref ActualPoints, ref FirstValidPoint, ref
InitialXOffset, ref InitialXTimeSeconds, ref InitialXTimeFraction, ref
XIncrement);
                }
                #endregion
            }
            catch (Exception ex)
            {
```

```
                Console.WriteLine(ex.Message);
            }
            finally
            {
                if (driver != null && driver.Initialized)
                {
                    #region Close Driver Instances
                    driver.Close();
                    Console.WriteLine("Driver Closed");
                    #endregion
                }
            }

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

```
  Multi Record

Driver Initialized

Configuring ActiveSource source to Software

Configuring Acquisition...
Number of records to acquire:    4
Number of record size:           16000
Rate of the sample clock:        20000000

Configuring Channel 1...
Voltage range:   16V
Coupling:        AC

Apply waveform source...

The hardware leaves the Idle state and waits for trigger to acquire waveform

Send first software trigger
Send second software trigger
Send third software trigger
Send fourth software trigger

Number of Acquired records:    4

Fetch Channel 1 all 4 acquired waveform record...
Fetch Channel 1 second and third acquired waveform record...
Driver Closed
Done - Press Enter to Exit
```

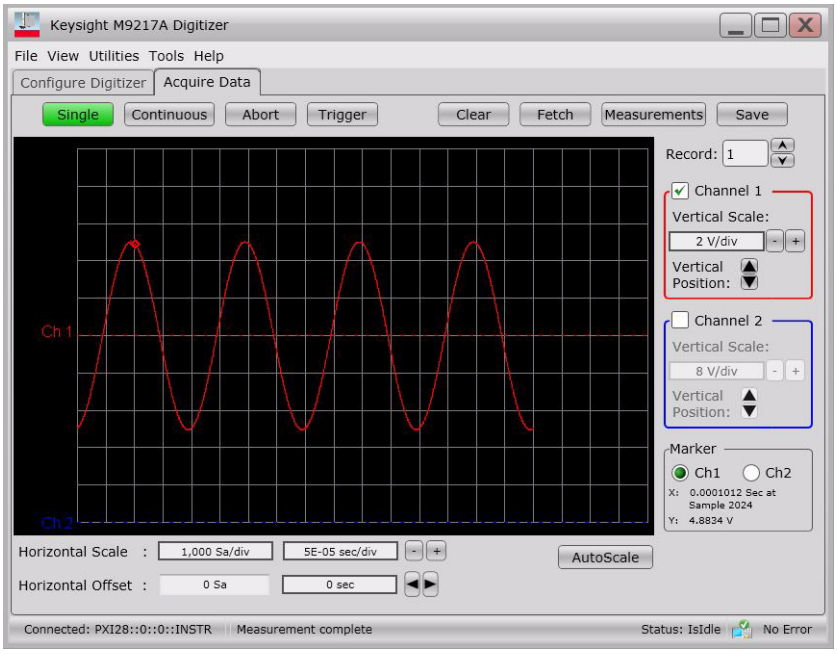


Figure 1-6 SFP first acquired waveform results with example program 5 settings

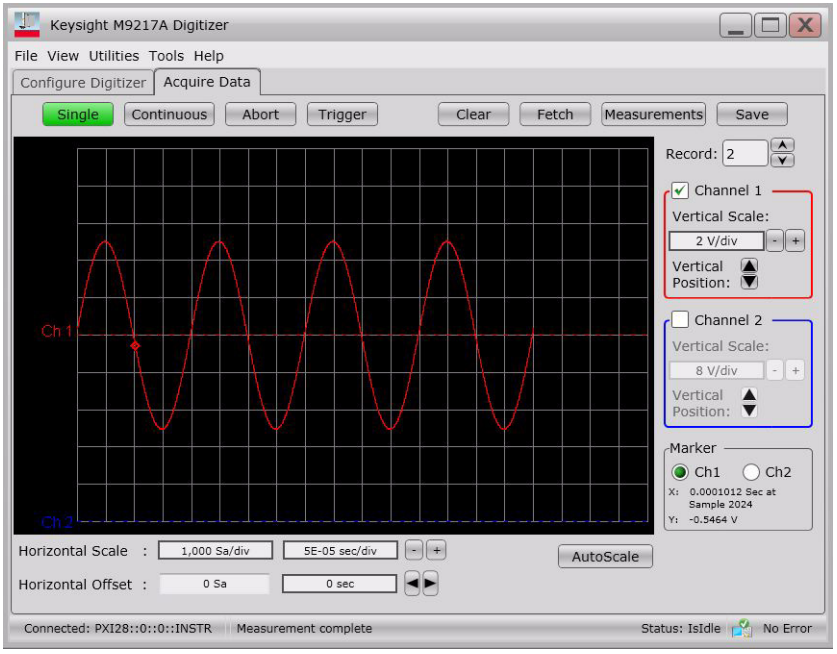


Figure 1-7 SFP second acquired waveform results with example program 5 settings

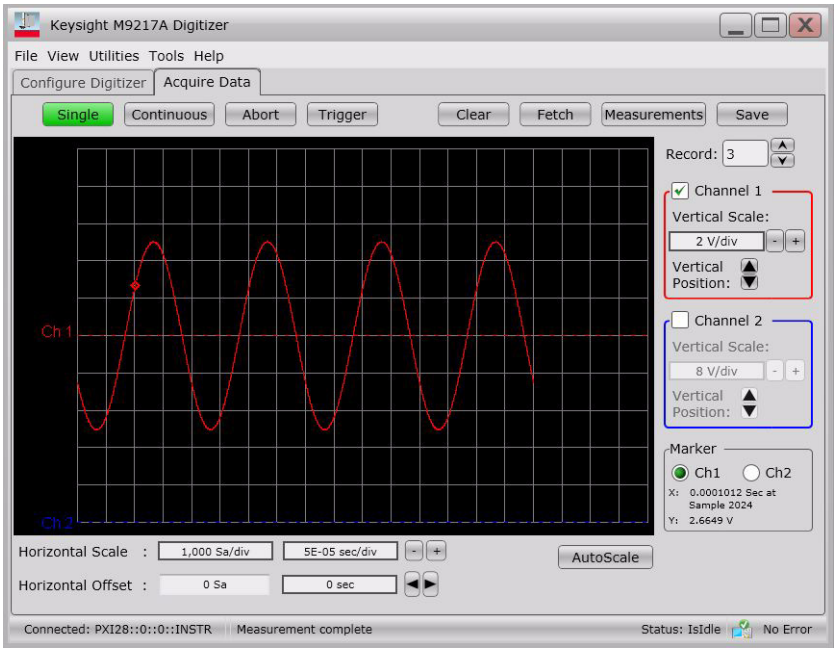


Figure 1-8 SFP third acquired waveform results with example program 5 settings

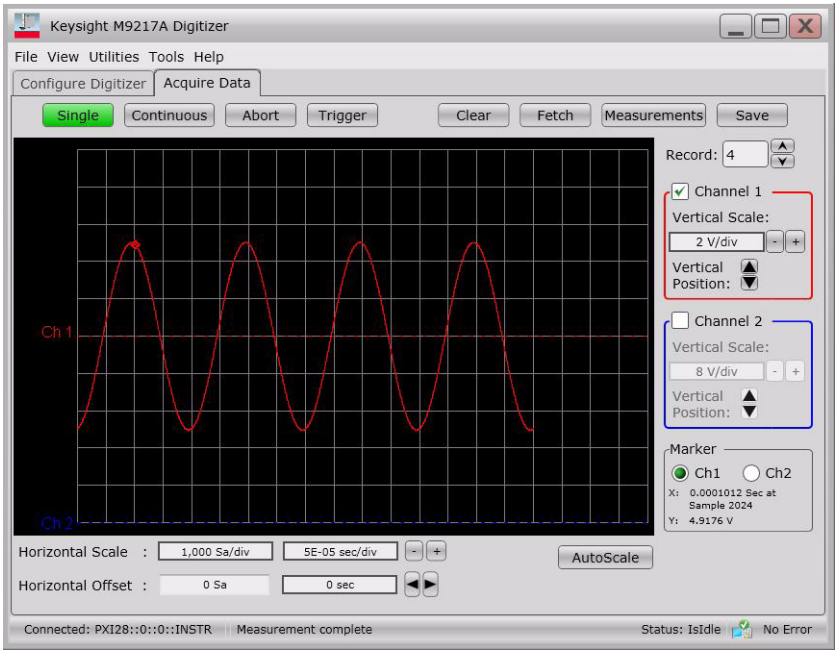


Figure 1-9 SFP fourth acquired waveform results with example program 5 settings

Glossary

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

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.

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.

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 IVI-Component drivers to be interchanged.

.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.

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. Keysight VISA-COM is used by its IVIComponent drivers and requires that Keysight VISA also be installed.

THIS PAGE HAS BEEN INTENTIONALLY LEFT BLANK.

Edition 1, October 1, 2015

M9217-90006
www.keysight.com

KEYSIGHT TECHNOLOGIES