

Keysight M9188A PXI D/A Converter 16-Bit, 0-30 V, 0-20 mA

NOTICE: This document contains references to Agilent Technologies. Agilent's former Test and Measurement business has become Keysight Technologies. For more information, go to www.keysight.com.



Notices

© Keysight Technologies 2014

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

Manual Part Number

M9188-90006

Edition

Edition 1, August 2014

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/M9188A
(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 change, without notice, in future editions. Further, to the maximum extent permitted by the applicable law, Keysight disclaims all warranties, either express or implied, with regard to this manual and any information contained herein, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. Keysight shall not be liable for errors or for incidental or consequential damages in connection with the furnishing, use, or performance of this document or of any information contained herein. Should Keysight and the user have a separate written agreement with warranty terms covering the material in this document that conflict with these terms, the warranty terms in the separate agreement shall control.

Technology 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 likes of 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 likes of 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 30V RMS, 42.4V peak, or 60VDC 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

- 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.
-

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 an Keysight office for information.

WARNING

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

Front and Rear Panels Symbols



The CE marking is the legal required labeling for several EU Directives of the European Union.

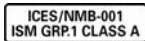
The CE marking shows that the product complies with all relevant European Legal Directives.



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 Inter-

ference-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.

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.

End of Life: 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.

THIS PAGE IS INTENTIONALLY LEFT BLANK.

Table of Contents

What You Will Learn in this Programming Guide	1
Related Websites	2
Related Documentation	3
Understanding the Overall Process Flow	3
Before Programming, Install Hardware, Software, and Licenses	4
Understanding the Application Programming Interfaces (API) for the M9188A PXI D/A Converter	6
IVI Instrument Classes (Defined by the IVI Foundation)	6
IVI Compliant or IVI Class Compliant	7
IVI Driver Types	8
IVI Driver Hierarchy	9
Instrument-Specific Hierarchies for the M9188A PXI D/A Converter	11
Naming Conventions Used to Program IVI Drivers	12
Tutorial: Create a Project with IVI-COM Using C#	14
Step 1 – Create a “Console Application”	14
Step 2 – Add References	15
Step 3 – Add using Statements	17
Step 4 – Create Instances of the IVI-COM Drivers	18
Step 5 – Initialize the Driver Instances	19
Step 6 – Write the Program Steps	24
Step 7 – Close the Driver	26
Step 8 – Building and Running a Complete Example Program Using Visual C#	26
Example Program 1: How to Print the Driver Properties and Close the Driver Sessions	27
Understanding and Working with the M9188A PXI D/A Converter	32
Product Overview	32
Output Mode	32

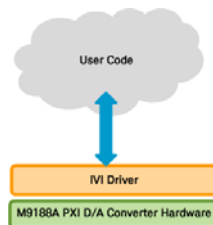
Operating Mode	33
Event	34
Trigger Out	35
Clock IN/OUT	36
Synchronizing Multi Modules	36
Example Program 2: How to Output a DC signal, a Sine Waveform, a Square Waveform, and a Triangle Waveform, Using the Software Trigger	38
Example Program 3: How to Output an Arbitrary Waveform and Create an Arbitrary Waveform from a File, Using the Software Trigger	45
Example Program 4: How to Output a DC Signal, Using the EXT Trigger	50
Example Program 5: How to Output an Event	54
Example Program 6: How to Output on Synchronization (Master/Slave)	58
IVI-COM and IVI-C API References	65
Unsupported IVI-COM APIs	65
Unsupported IVI-C APIs	73
Glossary	82

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 M9188A PXI D/A Converter



IVI-COM Console Applications that are covered in this programming guide are used to perform the basic functionality of the M9188A PXI D/A Converter.

The following basic functionality tests are covered:

- **Example Program 1:** How to Print Driver Properties and Close Driver Sessions
- **Example Program 2:** How to Output a DC Signal, a Sine Waveform, a Square Waveform, and a Triangle Waveform; Using the Software Trigger
- **Example Program 3:** How to Output an Arbitrary Waveform and Create an Arbitrary Waveform from a File; Using the Software Trigger
- **Example Program 4:** How to Output a DC Signal; Using the EXT Trigger
- **Example Program 5:** How to Output an Event
- **Example Program 6:** How to Output on Synchronization (Master/Slave)

Related Websites

- Keysight Technologies PXI and AXIe Modular Products (<http://www.keysight.com/find/modular>)
 - M9188A PXI D/A Converter (<http://www.keysight.com/find/M9188A>)
- Keysight Technologies (<http://www.keysight.com/>)
 - IVI Drivers and Components Downloads (<http://www.keysight.com/find/ivi>)
 - 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
<p>Startup Guide^[1]</p> <p>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.</p>	C:\Program Files\Keysight\M9188\Help
<p>Data Sheet^[1]</p> <p>In addition to a detailed product introduction, the data sheet supplies full product specifications.</p>	C:\Program Files\Keysight\M9188\Help
<p>LabVIEW Driver Reference (Online Help System)</p> <p>Provides detailed documentation of the LabVIEW G Driver API functions.</p>	C:\Program Files\Keysight\M9188\LabVIEW Driver Help

[1] If these links do not work, you can find these items at:

C:\Program Files\Keysight\M9188 or C:\Program Files(x86)\Keysight\M9188

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, defined in the Keysight M9188A PXI D/A Converter Startup Guide, M9188-90001, but repeated here must be completed before programmatically controlling the M9188A PXI D/A Converter 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 16.3.17914.4 or newer; this installation includes Keysight Connections Expert.
- Install the M9188A PXI D/A Converter 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/M9188A>
- 5 Install the hardware modules and make cable connections.

Before Programming, Install Hardware, Software, and Licenses

- 6 Verify operation of the modules (or the system that the modules create).

NOTE

Before programming or making measurements, conduct a Self-Test on the M9188A PXI D/A Converter to make sure there are no problems with the modules.

Once the software and hardware are installed and verification of operation has been performed, the M9188A is ready to be programmatically controlled.

Understanding the Application Programming Interfaces (API) for the M9188A PXI D/A Converter

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 M9188A PXI D/A Converter do not belong to any of these 13 IVI Instrument Classes and are therefore describes as “NoClass” modules.

IVI Instrument Classes (Defined by the IVI Foundation)

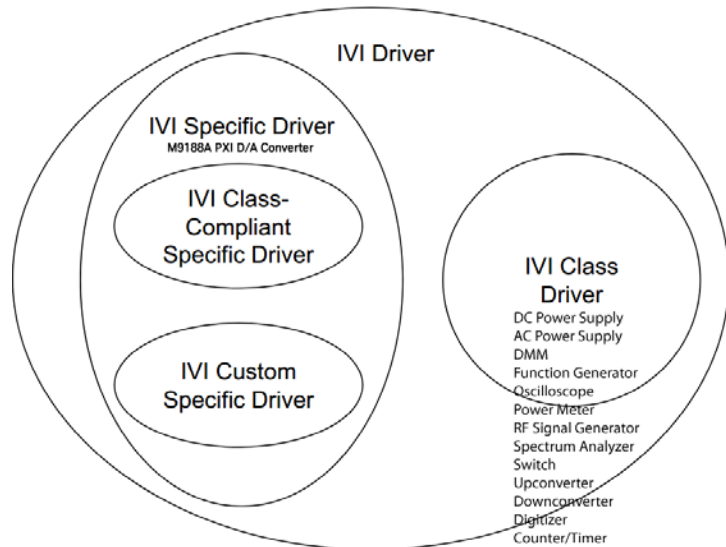
- DC Power Supply
- AC Power Supply
- DMM
- Function Generator
- Oscilloscope
- Power Meter
- RF Signal Generator
- Spectrum Analyzer
- Switch
- Upconverter
- Downconverter
- Digitizer
- Counter/Timer

IVI Compliant or IVI Class Compliant

The M9188A PXI D/A Converter are IVI Compliant, but not IVI Class Compliant; it does not belong to one of the 13 IVI Instrument Classes defined by the IVI Foundation.

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

IVI Driver Types



- **IVI Driver**
 - Implements the Inherent Capabilities Specification
 - Complies with all of the architecture specifications
 - May or may not comply with one of the 13 IVI Instrument Classes
 - Is either an IVI Specific Driver or an IVI Class Driver
- **IVI Specific Driver**
 - Is an IVI Driver that is written for a particular instrument such as the M9188A PXI D/A Converter
- **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.

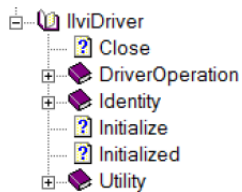
These interfaces are organized into two hierarchies: Class-Compliant Hierarchy and Instrument-Specific Hierarchy – and both include the IIVI driver interfaces.

- **Class-Compliant Hierarchy** - Since the M9188A PXI D/A Converter does not belong to one of the 13 IVI Classes, there is no Class-Compliant Hierarchy in its IVI Driver.
- **Instrument-Specific Hierarchy**
 - The M9188A PXI D/A Converter instrument-specific hierarchy has IKtM9188 at the root (where KtM9188 is the driver name).
IKtM9188 is the root interface and contains references to child interfaces, which in turn contain references to other child interfaces. Collectively, these interfaces define the instrument-specific hierarchy.
- The IIVI driver interfaces are incorporated into both hierarchies: Class-Compliant Hierarchy and Instrument-Specific Hierarchy.

The IIVI driver is the root interface for IVI Inherent Capabilities which are what the IVI Foundation has established as a set of functions and attributes that all IVI

Understanding the Application Programming Interfaces (API) for the M9188A PXI D/A Converter

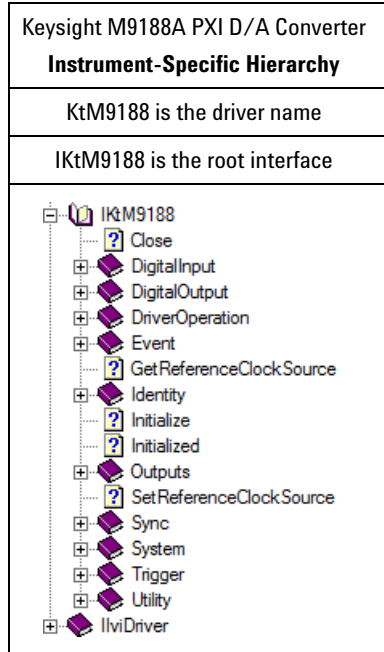
drivers must include – irrespective of which IVI instrument class the driver supports. These common functions and attributes are called IVI inherent capabilities and they are documented in IVI-3.2 – Inherent Capabilities Specification. Drivers that do not support any IVI instrument class such as the M9188A PXI D/A Converter must still include these IVI inherent capabilities.



IlviDriver
Close
DriverOperation
Identity
Initialize
Initialized
Utility

Instrument-Specific Hierarchies for the M9188A PXI D/A Converter

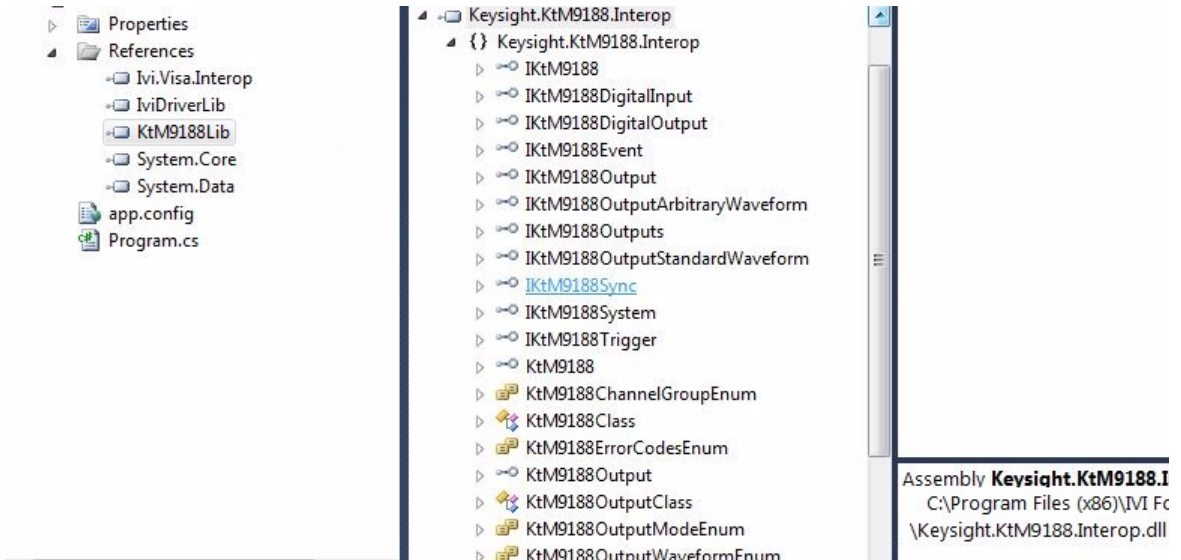
The following table lists the instrument-specific hierarchy interfaces for the M9188A PXI D/A Converter.



Understanding the Application Programming Interfaces (API) for the M9188A PXI D/A Converter

NOTE

To view interfaces available in the M9188A PXI D/A Converter, right-click the KtM9188Lib 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 “Ivi”
 - I<ClassName>
- Example: IviScope, IviDmm
 - Sub-interfaces add words to the base name that match the C hierarchy as close as possible
- Examples: IviFgenArbitrary, IviFgenArbitraryWaveform
- 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 “M9188Properties” 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.

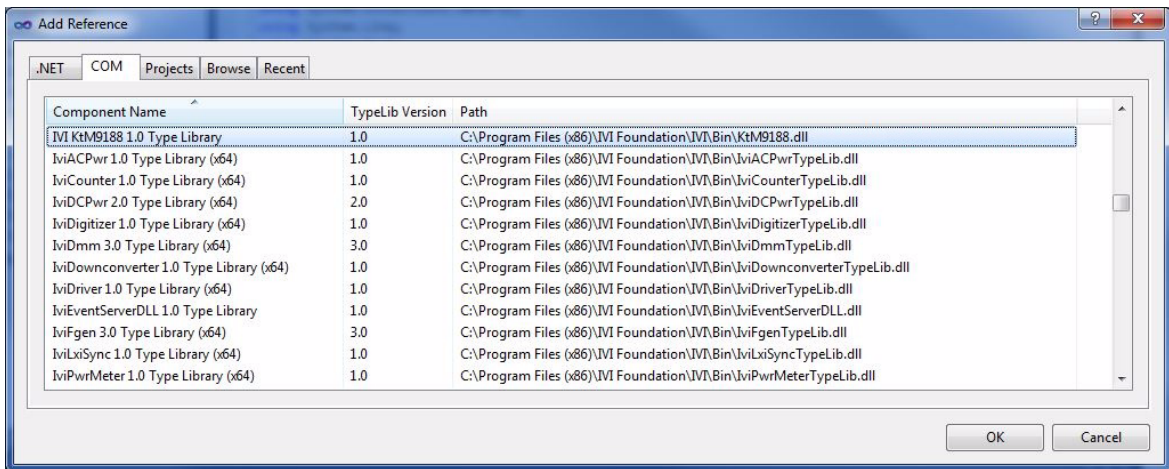
For this step, Solution Explorer must be visible (**View > Solution Explorer**) and the “Program.cs” editor window must be visible – select the **Program.cs** tab to bring it to the front view.

Step 2 – Add References

In order to access the M9188A PXI D/A Converter 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”.)

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



NOTE

- If you have not installed the IVI driver for the M9188A PXI D/A Converter (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 KtM9188 1.0 Type Library

NOTE

When any of the references for the M9188A PXI D/A Converter are added, the IviDriver 1.0 Type Library is also automatically added. This is visible as IviDriverLib under the project Reference; this reference houses the interface definitions for IVI inherent capabilities which are located in the file IviDriverTypeLib.dll (dynamically linked library).

- 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.

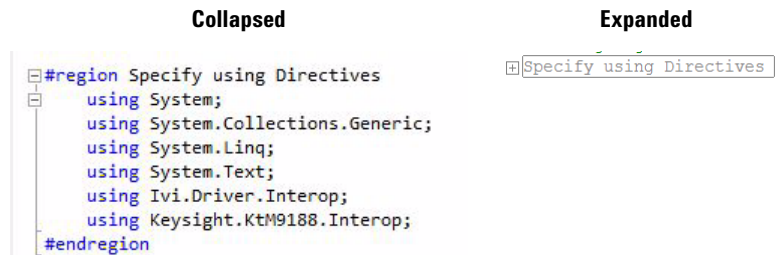
To access the IVI drivers without having to specify or type the full path of each interface or enum

These `using` statements should be added to your program:

```
using Ivi.Driver.Interop;  
using Keysight.KtM9188.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.



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 M9188A PXI D/A Converter 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 M9188A PXI D/A Converter IVI-COM driver uses direct instantiation.

To create driver instances

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

```
IKtM9188 driver = new KtM9188();
```

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

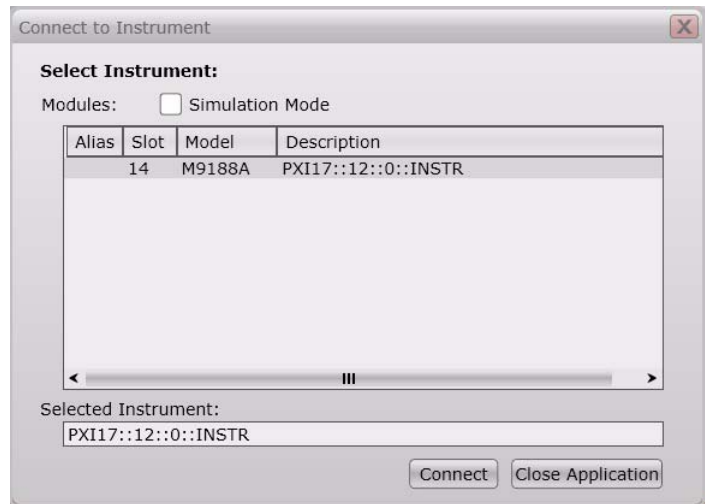
- If you are using Simulate Mode, you can set the Resource Name address string to:

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

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

Using the M9188A Soft Front Panel, you might get the following Resource Name address string.

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



Module Name	M9188A PXI D/A Converter
Slot Number	14
VISA Address	PXI17::12::0::INSTR

```
string resourceDesc = "PXI17::12::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 = "PXI20::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");
```

```
#region Initialize Driver Instances
driver = new KtM9188();

// Edit resource and options as needed. Resource is ignored if option Simulate=true
string resourceDesc = "PXI17::12::0::INSTR";

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

bool idquery = true;
bool reset = true;

// Initialize the driver
driver.Initialize(resourceDesc, idquery, reset, initOptions);
```

```
void IKtM9188.Initialize(string ResourceName, bool IdQuery, bool Reset, [string OptionString = null])
```

Opens the I/O session to the instrument. Driver methods and properties that access the instrument are only accessible after Initialize is called. Initialize optionally performs a Reset

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.

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

Property Type and Example Value	Description of Property
<pre>string ResourceName = PXI[bus]::device[::function][::INSTR] string ResourceName = "PXI17::12::0::INSTR";</pre>	<p>ResourceName</p> <p>The driver is typically initialized using a physical resource name descriptor, often a VISA resource descriptor.</p> <p>See the above procedure:</p> <p>"To determine the ResourceName"</p>
<pre>bool IdQuery = true;</pre>	<p>IdQuery</p> <p>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.</p>
<pre>bool Reset = true;</pre>	<p>Reset</p> <p>Setting Reset to true tells the driver to initially reset the instrument.</p>

Property Type and Example Value	Description of Property
<pre>string OptionString = "QueryInstrStatus=true, Simulate=true,</pre>	<p data-bbox="719 291 891 317">OptionString</p> <p data-bbox="719 328 1100 354">Setup the following initialization options:</p> <ul style="list-style-type: none"> <li data-bbox="719 366 1233 453">• QueryInstrStatus=true (Specifies whether the IVI specific driver queries the instrument status at the end of each user operation.) <li data-bbox="719 458 1272 569">• 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.) <li data-bbox="719 574 1179 661">• Cache=false (Specifies whether or not to cache the value of properties.) <li data-bbox="719 666 1208 753">• InterchangeCheck=false (Specifies whether the IVI specific driver performs interchangeability checking.) <li data-bbox="719 758 1208 845">• RangeCheck=false (Specifies whether the IVI specific driver validates attribute values and function parameters.) <li data-bbox="719 850 1255 951">• RecordCoercions=false (Specifies whether the IVI specific driver keeps a list of the value coercions it makes for ViInt32 and ViReal64 attributes.)
<pre>DriverSetup= Trace=false";</pre>	<ul style="list-style-type: none"> <li data-bbox="719 973 1272 1177">• 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.) <li data-bbox="719 1182 1158 1234">• Model=M9188A (Instrument model to use during simulation.) <li data-bbox="719 1239 1272 1321">• 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:

KtM9188A IVI Driver Reference

Start > All Programs > Keysight Instrument Drivers > IVI-COM-C Drivers > KtM9188 Dynamic DAC

Step 6 – Write the Program Steps

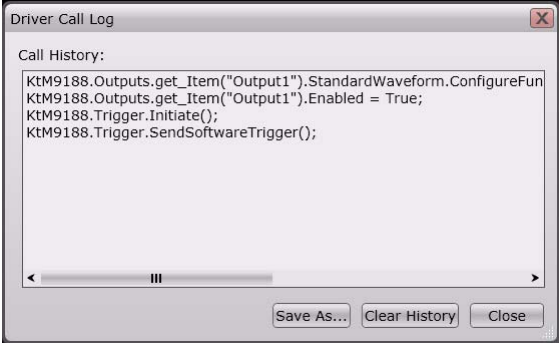
At this point, you can add program steps that use the driver instances to perform tasks.

Example: Using the Soft Front Panel to Write Program Commands

You may find it useful when developing a program to use the instrument's Soft Front Panel (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 soft front panel.

In this example, open the Soft Front Panel for the M9188A PXI D/A Converter and perform the following steps:

- 1** Set Output Channel 1 to DC Voltage mode
- 2** Enable Output Channel 1's output relay
- 3** Enable the output

Property Type and Example Value	Description of Property
<p>KtM9188 is the driver name used by the SFP.</p>	<p><code>driver</code> 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".</p> <pre>IKtM9188 driver = new KtM9188();</pre>
 <p>The screenshot shows a window titled "Driver Call Log" with a scrollable text area containing the following call history:</p> <pre>KtM9188.Outputs.get_Item("Output1").StandardWaveform.ConfigureFun KtM9188.Outputs.get_Item("Output1").Enabled = True; KtM9188.Trigger.Initiate(); KtM9188.Trigger.SendSoftwareTrigger();</pre> <p>At the bottom of the window are three buttons: "Save As...", "Clear History", and "Close".</p>	<pre>//Set the Output Channel 1 to DC Voltage mode with 10V level driver.Outputs.get_Item("Output1").Sta ndardWaveform.ConfigureFunction(KtM918 8OutputModeVoltage,KtM9188StdWaveformDC,10,0,0,1000,50); //Enable Output Channel 1?? output relay driver.Outputs.get_Item("Output1").Ena bled = True; //Enable the output driver.Trigger.Initiate(); driver.Trigger.SendSoftwareTrigger();</pre> <p>Or you could even configure channel with this:</p> <pre>//Set the Output Channel 1 to Voltage mode driver.Outputs.get_Item("Output1").Out putMode = KtM9188OutputModeEnum.KtM9188OutputMod eVoltage; //Set the Output Channel 1 to DC mode driver.Outputs.get_Item("Output1").Sta ndardWaveform.Function = KtM9188StdWaveformEnum.KtM9188StdWavef ormDC; //Set the Output Channel 1 to output 10V driver.Outputs.get_Item("Output1").Sta ndardWaveform.Amplitude = 10;</pre>

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\KtM9188\
Examples

or

C:\Program Files (x86)\IVI Foundation\IVI\Drivers\KtM9188\
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.

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

```
// Copy the following example code and compile it as a C# Console Application
// Example_KtM9188_PrintProperties.cs
Specify using Directives

namespace KtM9188_PrintProperties
{
    class Program
    {
        [STAThread]
        public static void Main(string[] args)
        {
            Console.WriteLine(" PrintProperties");
            Console.WriteLine();
            KtM9188 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_KtM9188_PrintProperties.cs
#region Specify using Directives
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Text;
using Keysight.KtM9188.Interop;
#endregion

namespace KtM9188_PrintProperties
{
    class Program
    {
        [STAThread]
        public static void Main(string[] args)
        {
            Console.WriteLine(" PrintProperties");
            Console.WriteLine();
            KtM9188 driver = null;

            try
            {
                #region Initialize Driver Instances
                driver = new KtM9188();

                // Edit resource and options as needed. Resource is ignored if option Simulate=true
                string resourceDesc = "PXI17::12::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);
            }
        }
    }
}

```

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

```
        #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();
}
}
```

```
PrintProperties
Driver Initialized
Identifier: KtM9188
Revision: 0.5.0.0
Vendor: Keysight Technologies
Description: IUI Driver for KtM9188
Model: M9188A
FirmwareRev: 11320314
Serial #: 781556
Simulate: False
Driver Closed
Done - Press Enter to Exit
```


NOTE

Disclaimer

© 2014 Keysight Technologies. All rights reserved.

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.

Understanding and Working with the M9188A PXI D/A Converter

Product Overview

The M9188A is a PXI based Dynamic Digital Analog Converter (Dynamic DAC). The module consists of four isolated banks. Each isolated bank has 4 DAC channels, resulting in a total of 16 DAC channels within one module.

Every DAC channel is capable of supplying voltages between 0 V DC and 30 V DC at a 16-bit resolution. Each channel can also generate current between 0 mA and +20 mA at a 16-bit resolution. Pre-defined waveforms (such as sine, square, and triangle) can be configured using the M9188A SFP or IVI driver. Every DAC channel also has its own 1 MSa memory for playing back arbitrary waveforms (user-defined waveforms) that are pre-downloaded from a host controller. This module is built into a single compact 3U PXI slot.

Output Mode

There are two main output modes – voltage and current.

Every channel can be configured to output voltage or current. All channels output can be enabled individually or simultaneously.

NOTE

You cannot change the settings (output mode, function, frequency, duty cycle, amplitude, offset, phase shift) on-the-fly. Channel output must be disabled and enabled again for new settings to take effect.

Output voltage range = 0 to 30 V;
Accuracy^[1] spec is + (0.1% + 5 mV).

Output current range = 0 to 20 mA;
Accuracy^[1] spec is + (0.15% + 10 μ A).

Operating Mode

There are two operating modes – Level (Static or DC) and Playback (Dynamic or AC).

- In LEVEL mode, every channel can be configured to output a static DC voltage or current. The output level can take effect immediately (software trigger) or when externally triggered (PXI, PXIStar or EXT trigger).
- In PLAYBACK mode, every channel can be configured to output a pre-defined waveform (sine, square, triangle) or arbitrary waveform (user define waveform pre-downloaded into memory through host PC).

When you configure a channel, you must select:

- 1 Voltage or Current output mode, and
- 2 Level or Playback mode

Using Level Mode

To use a channel to output DC, you need to decide whether it will be a voltage or current, and set the level. You can select voltage mode, and set any output level between 0 to +30 V. If you select current mode, output level can be between 0 to +20 mA.

Output voltage range = 0 to 30 V.

Output current range = 0 to 20mA.

Using Playback Mode

To use a channel to output waveform, you need to decide whether it will be a voltage or current waveform. This module can generate three standard waveforms: sine, square,

[1] For official accuracy and other specifications, kindly refer to M9188A Datasheet.

and triangle. You can also create your own custom waveform (arbitrary or user-defined) using module SFP. Size of the waveform cannot exceed 1 MSA.

There are two playback modes – Continuous or Burst. The default is continuous. The amplitude and offset cannot be set such that they combine to exceed the instrument's capability. Whichever one you set last in this situation will be modified to remain within the instrument's limits.

Continuous Mode

- Waveform data is first downloaded from the host controller to the module memory.
- Upon trigger reception, the module will update the DAC with the waveform memory data at a rate determined by the sample clock.
- At the end of the waveform memory, the module automatically loops back to the start of the waveform.
- The cycle will continue until the software commands it to stop (output disabled).

Burst Mode

- Waveform data is first downloaded from the host controller to the module memory.
- Upon trigger reception, the module will update the DAC with the waveform memory data at a rate determined by the sample clock.
- At the end of the waveform memory, the module automatically loops back to the start of the waveform.
- The output will stop when the Burst Count set is reached.

Event

- When EVENT is set to OFF – the module needs to be set to receive PXI Bus Trigger or external hardware trigger. In other words, Software is not a valid mode when EVENT is OFF. You will have to select either PXI Bus trigger or External hardware trigger.

- When EVENT is set to “EXT” – the module is set to send external trigger OUT (via Trigger I/O SMB) when any channel output is set to enabled. (In this condition, all channels in the module will automatically go into Software mode before setting the destination value (EXT). See more in the “Trigger Out” section.
- When EVENT is set to “PXI<n>” – the module is set to send external trigger OUT (via PXI Bus Trigger) when any channel output is set to enabled. (In this condition, all channels in the module will automatically go into Software mode before setting the destination value (EXT or PXI<n>). See more in the “Trigger Out” section.

Trigger Out

A Trigger Out signal is provided on the front panel trigger I/O SMB connector (EXT) or PXI Bus trigger. The output trigger pulse is TTL compatible and a standard width of 250ns (typical).

The module can be set to send trigger out pulse via EXT or PXI Trigger Bus.

- If EXT Trigger out is selected, a pulse is sent through the front panel SMB trigger I/O connector. The slope to respond to a trigger signal can be rising or falling edge.
- If PXI Bus trigger is selected, trigger signal will be sent through the PXI Trigger Bus to the specified destination module. There are eight unique trigger lines (PXI0, PXI1,...PXI7).

Clock IN/OUT

The external clock (via the EXT CLK SMB connector located at front panel), which is software programmable, allows either synchronization between two modules or to an external reference 10 MHz clock.

- By default, the EXT CLK is programmed to “EXT CLK IN”. At this state, the module's sampling clock frequency at EXT CLK IN will be set to 10 MHz.
- Master and Slave Devices - this is when you want to synchronize between two modules. One module is designated as the master and another one as the slave. The master will be programmed to EXT CLK OUT, while the slave to EXT CLK IN. By doing this, the slave module reference clock will be set to the 10 MHz clock signal from the master module. Since there is only one EXT CLK SMB connector on each module, only one slave module is possible.

There can be three selections of the 10 MHz system reference clock:

- Module internal clock (default)
- PXI clock
- External input clock

Synchronizing Multi Modules

Sample base clock for the M9188A is fixed to 2 MHz to achieve 2 MSa/s sampling rate as shown in **Figure 1** as the maximum allowed sampling rate for each channel is 500 kSa/s.

Understanding and Working with the M9188A PXI D/A Converter

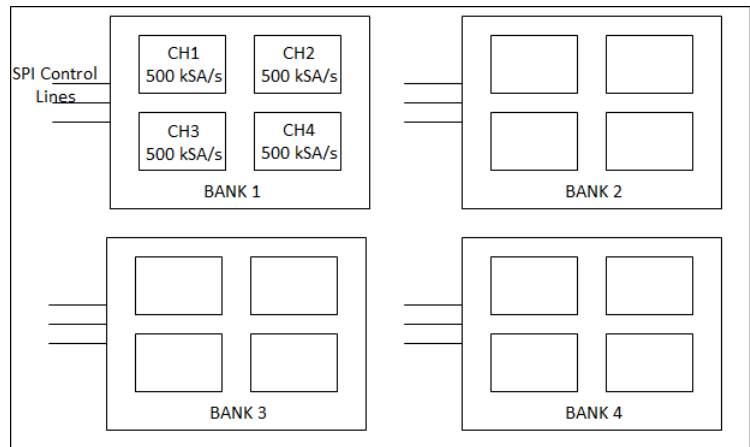


Figure 1 Maximum allowed sampling rate for each channel in every bank

The sample base clock is generated from the FPGA PLL using the 10 MHz reference clock and use in the module itself of send through the trigger lines to another module for multi module synchronization. **Figure 2** shows an example of a sample clock which is generated from the module or can be from other module through the trigger lines.

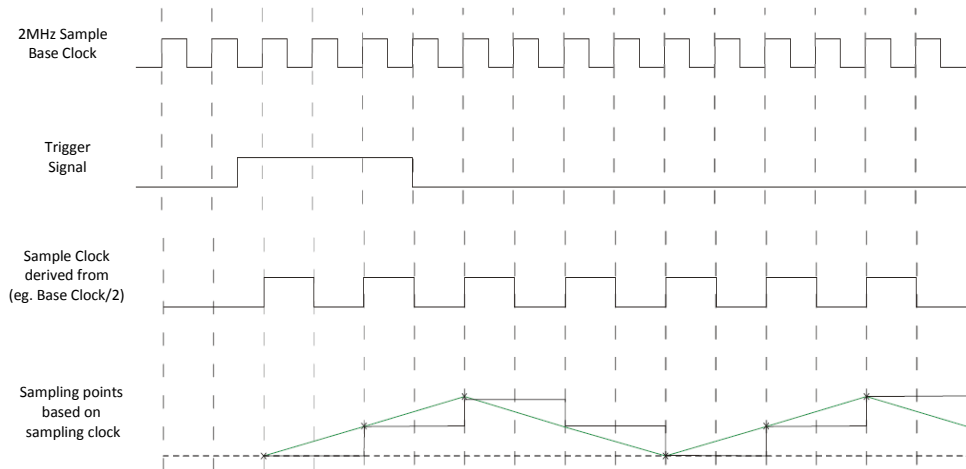


Figure 2 Example of synchronizing multiple modules with trigger signal and sample clock

To illustrate, a trigger signal is input from another trigger line and set to trigger on positive edge. As soon as a trigger signal is detected, sampling will start on the next clock cycle. In this example, the sampling clock is derived from base clock by a dividing factor of two. And output signal is shown by connecting the points of sampling.

Example Program 2: How to Output a DC signal, a Sine Waveform, a Square Waveform, and a Triangle Waveform, Using the Software Trigger

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

- 1 Apply changes to hardware
- 2 Output waveform

3 Report errors if any occur, and close the drivers

```

// Copy the following example code and compile it as a C# Console Application
// Example_KtM9188_StdWaveForm_SoftwareTrigger.cs
Specify using Directives

namespace KtM9188_StdWaveForm_SoftwareTrigger
{
    class Program
    {
        [STAThread]
        public static void Main(string[] args)
        {
            Console.WriteLine(" StdWaveForm_SoftwareTrigger");
            Console.WriteLine();
            KtM9188 driver = null;

            try
            {
                Initialize Driver Instances

                Software Trigger Settings

                Output Channel Settings - Output1 DC

                Output Channel Settings - Output2 Sine Wave

                Output Channel Settings - Output3 Square Wave

                Output Channel Settings - Output4 Triangle Wave

                Connect Output Relay

                Output On Software Trigger
            }
            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 Output a DC signal, a Sine Waveform, a Square Waveform, and a Triangle Waveform, Using the Software Trigger

- 1 Configure hardware Trigger Source to “Software”
- 2 Configure Output Channel 1 to DC mode
 - Voltage mode (10 Volt)
- 3 Configure Output Channel 2 to Sine Wave mode
 - Voltage mode (OffsetLevel 20 Volt and Amplitude 10 Volt)
 - 1 kHz
 - 0 degree phase shift
- 4 Configure Output Channel 3 to Square Wave mode
 - Voltage mode (OffsetLevel 15 Volt and Amplitude 3 Volt)
 - 500 Hz
 - 0 degree phase shift
 - 70% duty cycle
- 5 Configure Output Channel 4 to Triangle Wave mode
 - Current mode (OffsetLevel 0.01A and Amplitude 0.02A)
 - 2 kHz
 - 90 degree phase shift
- 6 Configure and turn on Output Channel relay
- 7 Output waveform

Example Program 2: How to Output a DC signal, a Sine Waveform, a Square Waveform, and a Triangle Waveform, Using the Software Trigger

```
// Copy the following example code and compile it as a C# Console Application
// Example_KtM9188_StdWaveForm_SoftwareTrigger.cs
#region Specify using Directives
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Text;
using Keysight.KtM9188.Interop;
#endregion
```

```

namespace KtM9188_StdWaveForm_SoftwareTrigger
{
    class Program
    {
        [STAThread]
        public static void Main(string[] args)
        {
            Console.WriteLine("  StdWaveForm_SoftwareTrigger");
            Console.WriteLine();
            KtM9188 driver = null;

            try
            {
                #region Initialize Driver Instances
                driver = new KtM9188();

                // Edit resource and options as needed. Resource is ignored if option
                Simulate=false
                string resourceDesc = "PXI20::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 Software Trigger Settings
                Console.WriteLine("Configuring trigger source to Software\n");
                driver.Trigger.Source = ("Software");
                #endregion

                #region Output Channel Settings - Output1 DC
                Console.WriteLine("Configuring Output1...");

                Console.WriteLine("Output mode:          Voltage");
                (driver.Outputs.get_Item("Output1")).OutputMode =
                KtM9188OutputModeEnum.KtM9188OutputModeVoltage;

                Console.WriteLine("Waveform function:  DC");
                (driver.Outputs.get_Item("Output1")).StandardWaveform.Function =
                KtM9188StdWaveformEnum.KtM9188StdWaveformDC;

                Console.WriteLine("DC amplitude:      10 Volts\n");
                (driver.Outputs.get_Item("Output1")).StandardWaveform.Amplitude = 10;
                #endregion
            }
        }
    }
}

```

Understanding and Working with the M9188A PXI D/A Converter

```
#region Output Channel Settings - Output2 Sine Wave
Console.WriteLine("Configuring Output2...");

Console.WriteLine("Output mode:      Voltage");
(driver.Outputs.get_Item("Output2")).OutputMode =
KtM9188OutputModeEnum.KtM9188OutputModeVoltage;

Console.WriteLine("Waveform function:  Sine");
(driver.Outputs.get_Item("Output2")).StandardWaveform.Function =
KtM9188StdWaveformEnum.KtM9188StdWaveformSine;

Console.WriteLine("Sine OffsetLevel:   20 Volts");
(driver.Outputs.get_Item("Output2")).StandardWaveform.OffsetLevel = 20;

Console.WriteLine("Sine Amplitude:     10 Volts");
(driver.Outputs.get_Item("Output2")).StandardWaveform.Amplitude = 10;

Console.WriteLine("Sine Frequency:     1KHz");
(driver.Outputs.get_Item("Output2")).StandardWaveform.Frequency = 1000;

Console.WriteLine("Sine PhaseShift:    0 Degree\n");
(driver.Outputs.get_Item("Output2")).StandardWaveform.PhaseShift = 0;
#endregion

#region Output Channel Settings - Output3 Square Wave
Console.WriteLine("Configuring Output3...");

Console.WriteLine("Output mode:      Voltage");
(driver.Outputs.get_Item("Output3")).OutputMode =
KtM9188OutputModeEnum.KtM9188OutputModeVoltage;

Console.WriteLine("Waveform function:  Square");
(driver.Outputs.get_Item("Output3")).StandardWaveform.Function =
KtM9188StdWaveformEnum.KtM9188StdWaveformSquare;

Console.WriteLine("Square OffsetLevel:  15 Volts");
(driver.Outputs.get_Item("Output3")).StandardWaveform.OffsetLevel = 15;

Console.WriteLine("Square Amplitude:   3 Volts");
(driver.Outputs.get_Item("Output3")).StandardWaveform.Amplitude = 3;

Console.WriteLine("Square Frequency:   500Hz");
(driver.Outputs.get_Item("Output3")).StandardWaveform.Frequency = 500;

Console.WriteLine("Square PhaseShift:  0 Degree");
(driver.Outputs.get_Item("Output3")).StandardWaveform.PhaseShift = 0;

Console.WriteLine("Square DutyCycle:   70 Percent\n");
(driver.Outputs.get_Item("Output3")).StandardWaveform.DutyCycleHigh = 70;
#endregion
```

```

#region Output Channel Settings - Output4 Triangle Wave

Console.WriteLine("Configuring Output4...");

Console.WriteLine("Output mode:           Current");
(driver.Outputs.get_Item("Output4")).OutputMode =
KtM9188OutputModeEnum.KtM9188OutputModeCurrent;

Console.WriteLine("Waveform function:       Triangle");
(driver.Outputs.get_Item("Output4")).StandardWaveform.Function =
KtM9188StdWaveformEnum.KtM9188StdWaveformTriangle;

Console.WriteLine("Triangle OffsetLevel:   0.01 A");
(driver.Outputs.get_Item("Output4")).StandardWaveform.OffsetLevel = 0.01;

Console.WriteLine("Triangle Amplitude:      0.02 A");
(driver.Outputs.get_Item("Output4")).StandardWaveform.Amplitude = 0.02;

Console.WriteLine("Triangle Frequency:     2KHz");
(driver.Outputs.get_Item("Output4")).StandardWaveform.Frequency = 2000;

Console.WriteLine("Triangle PhaseShift:    90 Degree\n");
(driver.Outputs.get_Item("Output4")).StandardWaveform.PhaseShift = 90;
#endregion

#region Connect Output Relay
Console.WriteLine("Enabling Output1 output relay");
(driver.Outputs.get_Item("Output1")).Enabled = true;

Console.WriteLine("Enabling Output2 output relay");
(driver.Outputs.get_Item("Output2")).Enabled = true;

Console.WriteLine("Enabling Output3 output relay");
(driver.Outputs.get_Item("Output3")).Enabled = true;

Console.WriteLine("Enabling Output4 output relay\n");
(driver.Outputs.get_Item("Output4")).Enabled = true;
#endregion

#region Output On Software Trigger
Console.WriteLine("Put hardware into wait for trigger state\n");
driver.Trigger.Initiate();

Console.WriteLine("Send software trigger to hardware\n");
driver.Trigger.SendSoftwareTrigger();
#endregion
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
finally

```

Understanding and Working with the M9188A PXI D/A Converter

```
{
    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();
}
}
```

```
StdWaveForm_SoftwareTrigger
Driver Initialized
Configuring trigger source to Software
Configuring Output1...
Output mode:      Voltage
Waveform function: DC
DC amplitude:     10 Volts
Configuring Output2...
Output mode:      Voltage
Waveform function: Sine
Sine OffsetLevel: 20 Volts
Sine Amplitude:   10 Volts
Sine Frequency:   1KHz
Sine PhaseShift:  0 Degree
Configuring Output3...
Output mode:      Voltage
Waveform function: Square
Square OffsetLevel: 15 Volts
Square Amplitude: 3 Volts
Square Frequency: 500Hz
Square PhaseShift: 0 Degree
Square DutyCycle: 70 Percent
Configuring Output4...
Output mode:      Current
Waveform function: Triangle
Triangle OffsetLevel: 0.01 A
Triangle Amplitude: 0.02 A
Triangle Frequency: 2KHz
Triangle PhaseShift: 90 Degree
Enabling Output1 output relay
Enabling Output2 output relay
Enabling Output3 output relay
Enabling Output4 output relay
Put hardware into wait for trigger state
Send software trigger to hardware
Driver Closed
Done - Press Enter to Exit
```

Example Program 3: How to Output an Arbitrary Waveform and Create an Arbitrary Waveform from a File, Using the Software Trigger

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

- 1** Apply changes to hardware
- 2** Output waveform
- 3** Report errors if any occur, and close the drivers

Understanding and Working with the M9188A PXI D/A Converter

```
// Copy the following example code and compile it as a C# Console Application
// Example_KtM9188_ArbWaveForm_SoftwareTrigger.cs
Specify using Directives

namespace KtM9188_ArbWaveForm_SoftwareTrigger
{
    class Program
    {
        [STAThread]
        public static void Main(string[] args)
        {
            Console.WriteLine(" ArbWaveForm_SoftwareTrigger");
            Console.WriteLine();
            KtM9188 driver = null;

            try
            {
                Initialize Driver Instances

                Software Trigger Settings

                Output Channel Settings - Output5 ArbCreate

                Output Channel Settings - Output6 ArbCreateFromFile

                Connect Output Relay

                Output On Software Trigger

            }
            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 Output an Arbitrary Waveform and Create an Arbitrary Waveform from a File, Using the Software Trigger

- 1 Configure hardware Trigger Source to “Software”
- 2 Creating 30 V peak-to-peak arbitrary sine waveform and configure it to Output Channel 5
- 3 Load custom arbitrary waveform from file (.csv) to Output Channel 6
- 4 Configure and turn on Output Channel relay
- 5 Output waveform

Example Program 3: How to Output an Arbitrary Waveform and Create an Arbitrary Waveform from a File, Using the Software Trigger

```
// Copy the following example code and compile it as a C# Console Application
// Example_KtM9188_ArbWaveForm_SoftwareTrigger.cs
#region Specify using Directives
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Text;
using Keysight.KtM9188.Interop;
#endregion

namespace KtM9188_ArbWaveForm_SoftwareTrigger
{
    class Program
    {
        [STAThread]
        public static void Main(string[] args)
        {
            Console.WriteLine(" ArbWaveForm_SoftwareTrigger");
            Console.WriteLine();
            KtM9188 driver = null;

            try
            {
                #region Initialize Driver Instances
                driver = new KtM9188();

                // Edit resource and options as needed. Resource is ignored if option
                Simulate=true
                string resourceDesc = "PXI20::0::0::INSTR";
```

Understanding and Working with the M9188A PXI D/A Converter

```
        string initOptions = "QueryInstrStatus=true, Simulate=true, 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 Software Trigger Settings
        Console.WriteLine("Configuring trigger source to Software\n");
        driver.Trigger.Source = ("Software");
        #endregion

        #region Output Channel Settings - Output5 ArbCreate

        double[] ArbData = new double[1500];

        Console.WriteLine("Configuring Output5...");

        Console.WriteLine("Output mode:          Voltage");
        (driver.Outputs.get_Item("Output5")).OutputMode =
KtM9188OutputModeEnum.KtM9188OutputModeVoltage;

        // Compute peak to peak 30V Sine Wave
        for (int i = 0; i < 1500; i++)
        {
            ArbData[i] = 15 + (15 * Math.Sin(2 * Math.PI * ((double)i / 1500)));
        }

        Console.WriteLine("Configuring Arbitrary data\n");
        (driver.Outputs.get_Item("Output5")).ArbitraryWaveform.Create(ref
ArbData);

        #endregion

        #region Output Channel Settings - Output6 ArbCreateFromFile
        Console.WriteLine("Configuring Output6...");

        Console.WriteLine("Output mode:          Voltage");
        (driver.Outputs.get_Item("Output6")).OutputMode =
KtM9188OutputModeEnum.KtM9188OutputModeVoltage;
```

```

        // Configuring Arbitrary data
        Console.WriteLine("Configuring Arbitrary data from file \
CombineWaveVoltage_8500.csv\\n");

(driver.Outputs.get_Item("Output6")).ArbitraryWaveform.CreateFromFile("CombineWaveVoltage_8500.csv", 8500);
        #endregion

        #region Connect Output Relay
        Console.WriteLine("Enabling Output5 output relay");
        (driver.Outputs.get_Item("Output5")).Enabled = true;

        Console.WriteLine("Enabling Output6 output relay\\n");
        (driver.Outputs.get_Item("Output6")).Enabled = true;
        #endregion

        #region Output On Software Trigger
        Console.WriteLine("Put hardware into wait for trigger state\\n");
        driver.Trigger.Initiate();

        Console.WriteLine("Send software trigger to hardware\\n");
        driver.Trigger.SendSoftwareTrigger();
        #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();
}
}
}

```

```
ArbWaveForm_SoftwareTrigger
Driver Initialized
Configuring trigger source to Software
Configuring Output5...
Output mode:      Voltage
Configuring Arbitrary data
Configuring Output6...
Output mode:      Voltage
Configuring Arbitrary data from file "CombineWaveVoltage_8500.csv"
Enabling Output5 output relay
Enabling Output6 output relay
Put hardware into wait for trigger state
Send software trigger to hardware
Driver Closed
Done - Press Enter to Exit
```

Example Program 4: How to Output a DC Signal, Using the EXT Trigger

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

- 1 Apply changes to hardware
- 2 Wait for trigger to output DC
- 3 Report errors if any occur, and close the drivers

Understanding and Working with the M9188A PXI D/A Converter

```
// Copy the following example code and compile it as a C# Console Application
// Example_KtM9188_DC_EXTTrigger.cs
Specify using Directives

namespace KtM9188_DC_EXTTrigger
{
    class Program
    {
        [STAThread]
        public static void Main(string[] args)
        {
            Console.WriteLine(" DC_EXTTrigger");
            Console.WriteLine();
            KtM9188 driver = null;

            try
            {
                Initialize Driver Instances

                EXT Trigger Settings

                Output Channel Settings

                Connect Output Relay

                Wait for trigger
            }
            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 Output a DC Signal, Using the EXT Trigger

- 1 Configure hardware Trigger Source to “EXT”
- 2 Output Channel 1 to DC mode
 - Voltage mode (10 Volt)

Understanding and Working with the M9188A PXI D/A Converter

- 3 Configure and turn on Output Channel relay
- 4 Wait for trigger to output DC

Example Program 4: How to Output a DC Signal, Using the EXT Trigger

```
// Copy the following example code and compile it as a C# Console Application
// Example_KtM9188_DC_EXTTrigger.cs
#region Specify using Directives
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Text;
using Keysight.KtM9188.Interop;
#endregion

namespace KtM9188_DC_EXTTrigger
{
    class Program
    {
        [STAThread]
        public static void Main(string[] args)
        {
            Console.WriteLine(" DC_EXTTrigger");
            Console.WriteLine();
            KtM9188 driver = null;

            try
            {
                #region Initialize Driver Instances
                driver = new KtM9188();

                // Edit resource and options as needed. Resource is ignored if option
                Simulate=true
                string resourceDesc = "PXI20::0::0::INSTR";

                string initOptions = "QueryInstrStatus=true, Simulate=true, 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 EXT Trigger Settings
                Console.WriteLine("Configuring trigger source to EXT\n");
                driver.Trigger.Configure("EXT",
                KtM9188TriggerSlopeEnum.KtM9188TriggerSlopePositive);
            }
        }
    }
}
```

```

        #endregion

        #region Output Channel Settings
        Console.WriteLine("Configuring Output1...");

        Console.WriteLine("Output mode:      Voltage");
        (driver.Outputs.get_Item("Output1")).OutputMode =
KtM9188OutputModeEnum.KtM9188OutputModeVoltage;

        Console.WriteLine("Waveform function:  DC");
        (driver.Outputs.get_Item("Output1")).StandardWaveform.Function =
KtM9188StdWaveformEnum.KtM9188StdWaveformDC;

        Console.WriteLine("DC amplitude:      10 Volts\n");
        (driver.Outputs.get_Item("Output1")).StandardWaveform.Amplitude = 10;
        #endregion

        #region Connect Output Relay
        Console.WriteLine("Enabling Output1 output relay\n");
        (driver.Outputs.get_Item("Output1")).Enabled = true;
        #endregion

        #region Wait for trigger
        Console.WriteLine("Put hardware into wait for trigger state\n");
        driver.Trigger.Initiate();
        #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();
}
}
}

```

```
DC_EXTTrigger
Driver Initialized
Configuring trigger source to EXT
Configuring Output1...
Output mode:      Voltage
Waveform function: DC
DC amplitude:     10 Volts
Enabling Output1 output relay
Put hardware into wait for trigger state
Driver Closed
Done - Press Enter to Exit
```

Example Program 5: How to Output an Event

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

- 1 Apply changes to hardware
- 2 Output Event output pulse
- 3 Report errors if any occur, and close the drivers

Understanding and Working with the M9188A PXI D/A Converter

```
// Copy the following example code and compile it as a C# Console Application
// Example_KtM9188_EVENT.cs
Specify using Directives

namespace KtM9188_EVENT
{
    class Program
    {
        [STAThread]
        public static void Main(string[] args)
        {
            Console.WriteLine(" DC_EVENT");
            Console.WriteLine();
            KtM9188 driver = null;

            try
            {
                Initialize Driver Instances

                Software Trigger Settings

                Event Destination Settings

                Output Event Output Pulse
            }
            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 Output an Event

- 1 Configure hardware Trigger Source to “Software”
- 2 Configure hardware Event destination to “EXT”
- 3 Output Event output pulse

Example Program 5: How to Output an Event

```
// Copy the following example code and compile it as a C# Console Application
// Example_KtM9188_EVENT.cs
#region Specify using Directives
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Text;
using Keysight.KtM9188.Interop;
#endregion

namespace KtM9188_EVENT
{
    class Program
    {
        [STAThread]
        public static void Main(string[] args)
        {
            Console.WriteLine(" DC_EVENT");
            Console.WriteLine();
            KtM9188 driver = null;

            try
            {
                #region Initialize Driver Instances
                driver = new KtM9188();

                // Edit resource and options as needed. Resource is ignored if option
                Simulate=true
                string resourceDesc = "PXI20::0::0::INSTR";

                string initOptions = "QueryInstrStatus=true, Simulate=true, 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 Software Trigger Settings
Console.WriteLine("Configuring trigger source to Software\n");
driver.Trigger.Source = ("Software");
#endregion

#region Event Destination Settings
Console.WriteLine("Configuring event Destination to EXT\n");
driver.Event.Destination = ("EXT");
#endregion

#region Output Event Output Pulse
Console.WriteLine("Send out Event output pulse\n");
driver.Trigger.SendSoftwareTrigger();
#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();
}
}
}

```

```
DC_EUENT
Driver Initialized
Configuring trigger source to Software
Configuring event Destination to EXT
Send out Event output pulse
Driver Closed
Done - Press Enter to Exit
```

Example Program 6: How to Output on Synchronization (Master/Slave)

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

- 1 Apply changes to Master and Slave hardware
- 2 Output on Synchronization
- 3 Report errors if any occur, and close the drivers

```

// Copy the following example code and compile it as a C# Console Application
// Example_KtM9188_Synchronization.cs
Specify using Directives

namespace KtM9188_Synchronization
{
    class Program
    {
        [STAThread]
        public static void Main(string[] args)
        {
            Console.WriteLine(" Synchronization");
            Console.WriteLine();
            KtM9188 MasterDriver = null;
            KtM9188 SlaveDriver = null;

            try
            {
                Initialize Driver Instances
                Trigger Settings
                Master Event Destination Settings
                Master Slave Settings
                Output Channel Settings - Master Output1
                Output Channel Settings - Slave Output1
                Connect Output Relay
                Wait For Trigger
                Master Software Trigger and Event out
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
            finally
            {
                if (MasterDriver != null && MasterDriver.Initialized)
                {
                    Close Driver Instances
                }
                if (SlaveDriver != null && SlaveDriver.Initialized)
                {
                    Close Driver Instances
                }
            }
            Console.WriteLine("Done - Press Enter to Exit");
            Console.ReadLine();
        }
    }
}

```

Pseudo-code of How to Output on Synchronization (Master/Slave)

- 1 Configure hardware Trigger Source:
 - “Software” for Master hardware
 - “PXI1” for Slave hardware
- 2 Configure Master hardware Event destination to “PXI1”
- 3 Configure Master hardware as Sync Master at “PXI0” and Slave hardware as Sync Slave at “PXI0”
- 4 Configure Master Output Channel 1 to Sine Wave mode
 - Voltage mode (OffsetLevel 20 Volt and Amplitude 10 Volt)
 - 1 kHz
 - 0 degree phase shift
- 5 Configure Slave Output Channel 1 to Sine Wave mode
 - Voltage mode (OffsetLevel 2 Volt and Amplitude 1 Volt)
 - 1 kHz
 - 0 degree phase shift
- 6 Configure and turn on Output Channel relay
- 7 Apply wait for trigger for Master and Slave hardware
- 8 Configure Master hardware to Send software trigger and output Event output pulse to PXI1

Example Program 6: How to Output Event (Master/Slave)

```
// Copy the following example code and compile it as a C# Console Application
// Example_KtM9188_Synchronization.cs
#region Specify using Directives
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Text;
using Keysight.KtM9188.Interop;
#endregion

namespace KtM9188_Synchronization
{
    class Program
    {
        [STAThread]
        public static void Main(string[] args)
        {
            Console.WriteLine(" Synchronization");
        }
    }
}
```

```

Console.WriteLine();
KtM9188 MasterDriver = null;
KtM9188 SlaveDriver = null;

try
{
    #region Initialize Driver Instances
    MasterDriver = new KtM9188();
    SlaveDriver = new KtM9188();

    // Edit resource and options as needed. Resource is ignored if option
    Simulate=true
    string MasterResourceDesc = "PXI20::0::0::INSTR";
    string SlaveResourceDesc = "PXI20::10::0::INSTR";

    string initOptions = "QueryInstrStatus=true, Simulate=true, 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
    MasterDriver.Initialize(MasterResourceDesc, idquery, reset, initOptions);
    Console.WriteLine("Master Driver Initialized");

    SlaveDriver.Initialize(SlaveResourceDesc, idquery, reset, initOptions);
    Console.WriteLine("Slave Driver Initialized\n");
    #endregion

    #region Trigger Settings
    Console.WriteLine("Configuring Master trigger source to Software");
    MasterDriver.Trigger.Source = ("Software");

    Console.WriteLine("Configuring Slave trigger source to PXI1\n");
    SlaveDriver.Trigger.Configure("PXI1",
    KtM9188TriggerSlopeEnum.KtM9188TriggerSlopePositive);
    #endregion

    #region Master Event Destination Settings
    Console.WriteLine("Configuring Master Event Destination to PXI1\n");
    MasterDriver.Event.Configure("PXI1");
    #endregion

    #region Master Slave Settings

    Console.WriteLine("Configuring Master's Sync mode to Master mode at PXI0");
    MasterDriver.Sync.Configure(KtM9188SyncModeEnum.KtM9188SyncModeMaster,
    "PXI0", true);

    Console.WriteLine("Configuring Slave's Sync mode to Slave mode at PXI0\n");
    SlaveDriver.Sync.Configure(KtM9188SyncModeEnum.KtM9188SyncModeSlave,
    "PXI0", true);
    #endregion

    #region Output Channel Settings - Master Output1
    Console.WriteLine("Configuring Master Output1...");

    Console.WriteLine("Output mode:          Voltage");

```

Understanding and Working with the M9188A PXI D/A Converter

```
(MasterDriver.Outputs.get_Item("Output1")).OutputMode =
KtM9188OutputModeEnum.KtM9188OutputModeVoltage;

    Console.WriteLine("Waveform function:  Sine");
    (MasterDriver.Outputs.get_Item("Output1")).StandardWaveform.Function =
KtM9188StdWaveformEnum.KtM9188StdWaveformSine;

    Console.WriteLine("Sine OffsetLevel:    20 Volts");
    (MasterDriver.Outputs.get_Item("Output1")).StandardWaveform.OffsetLevel =
20;

    Console.WriteLine("Sine Amplitude:      10 Volts");
    (MasterDriver.Outputs.get_Item("Output1")).StandardWaveform.Amplitude =
10;

    Console.WriteLine("Sine Frequency:      1KHz");
    (MasterDriver.Outputs.get_Item("Output1")).StandardWaveform.Frequency =
1000;

    Console.WriteLine("Sine PhaseShift:    0 Degree\n");
    (MasterDriver.Outputs.get_Item("Output1")).StandardWaveform.PhaseShift =
0;
#endregion

#region Output Channel Settings - Slave Output1
    Console.WriteLine("Configuring Slave Output1...");

    Console.WriteLine("Output mode:      Voltage");
    (SlaveDriver.Outputs.get_Item("Output1")).OutputMode =
KtM9188OutputModeEnum.KtM9188OutputModeVoltage;

    Console.WriteLine("Waveform function:  Sine");
    (SlaveDriver.Outputs.get_Item("Output1")).StandardWaveform.Function =
KtM9188StdWaveformEnum.KtM9188StdWaveformSine;

    Console.WriteLine("Sine OffsetLevel:    2 Volts");
    (SlaveDriver.Outputs.get_Item("Output1")).StandardWaveform.OffsetLevel =
2;

    Console.WriteLine("Sine Amplitude:      1 Volts");
    (SlaveDriver.Outputs.get_Item("Output1")).StandardWaveform.Amplitude = 1;

    Console.WriteLine("Sine Frequency:      1KHz");
    (SlaveDriver.Outputs.get_Item("Output1")).StandardWaveform.Frequency =
1000;

    Console.WriteLine("Sine PhaseShift:    0 Degree\n");
    (SlaveDriver.Outputs.get_Item("Output1")).StandardWaveform.PhaseShift = 0;
#endregion

#region Connect Output Relay
    Console.WriteLine("Enabling Master Output1 output relay");
    (MasterDriver.Outputs.get_Item("Output1")).Enabled = true;

    Console.WriteLine("Enabling Slave Output1 output relay\n");
    (SlaveDriver.Outputs.get_Item("Output1")).Enabled = true;
#endregion

#region Wait For Trigger
    Console.WriteLine("Put Master hardware into wait for trigger state");
    MasterDriver.Trigger.Initiate();
```



```

        Console.WriteLine("Put Slave hardware into wait for trigger state\n");
        SlaveDriver.Trigger.Initiate();
        #endregion

        #region Master Software Trigger and Event out
        Console.WriteLine("Master hardware send Software trigger and send out Event
PXI1\n");
        MasterDriver.Trigger.SendSoftwareTrigger();
        #endregion

    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
    finally
    {
        if (MasterDriver != null && MasterDriver.Initialized)
        {
            #region Close Driver Instances
            MasterDriver.Close();
            Console.WriteLine("Master Driver Closed");
            #endregion
        }
        if (SlaveDriver != null && SlaveDriver.Initialized)
        {
            #region Close Driver Instances
            SlaveDriver.Close();
            Console.WriteLine("Slave Driver Closed");
            #endregion
        }
    }
    Console.WriteLine("Done - Press Enter to Exit");
    Console.ReadLine();
}
}
}
}
}
}

```

Understanding and Working with the M9188A PXI D/A Converter

```
Synchronization
Master Driver Initialized
Slave Driver Initialized

Configuring Master trigger source to Software
Configuring Slave trigger source to PXI1

Configuring Master Event Destination to PXI1

Configuring Master's Sync mode to Master mode at PXI0
Configuring Slave's Sync mode to Slave mode at PXI0

Configuring Master Output1...
Output mode:      Voltage
Waveform function: Sine
Sine OffsetLevel: 20 Volts
Sine Amplitude:  10 Volts
Sine Frequency:  1KHz
Sine PhaseShift: 0 Degree

Configuring Slave Output1...
Output mode:      Voltage
Waveform function: Sine
Sine OffsetLevel: 2 Volts
Sine Amplitude:  1 Volts
Sine Frequency:  1KHz
Sine PhaseShift: 0 Degree

Enabling Master Output1 output relay
Enabling Slave Output1 output relay

Put Master hardware into wait for trigger state
Put Slave hardware into wait for trigger state

Master hardware send Software trigger and send out Event PXI1

Master Driver Closed
Slave Driver Closed
Done - Press Enter to Exit
```

IVI-COM and IVI-C API References

For a list of all IVI-COM and IVI-C APIs please refer to the *KtM9188A IVI Driver Reference Help File*.

The *KtM9188A IVI Driver Reference Help File* can be found in **Start > All Programs > Keysight Instrument Drivers > IVI-COM-C Drivers > KtM9188 Dynamic DAC**

Unsupported IVI-COM APIs

The IVI-COM APIs that are not supported are shown below.

KtM9188 IVI Driver Reference

Keysight Technologies

IIVIvDriverOperation.ClearInterchangeWarnings Method

-NOT SUPPORTED- Clears the list of interchangeability warnings that the IVI specific driver maintains.

Namespace: Ivi.Driver.Interop

Assembly: Ivi.Driver.Interop (in Ivi.Driver.Interop.DLL)

Syntax

Visual Basic

```
Public Sub ClearInterchangeWarnings ( _
)
```

C#

```
public void ClearInterchangeWarnings(
)
```

Visual C++

```
HRESULT ClearInterchangeWarnings (
);
```

IviDriverOperation.GetNextInterchangeWarning Method

-NOT SUPPORTED- Returns the oldest warning from the interchange warning list. Records are only added to the list if InterchangeCheck is True.

Namespace: Ivi.Driver.Interop

Assembly: Ivi.Driver.Interop (in Ivi.Driver.Interop.DLL)

Syntax

Visual Basic

```
Public Function GetNextInterchangeWarning ( _  
) As String
```

C#

```
public string GetNextInterchangeWarning(  
)
```

Visual C++

```
HRESULT GetNextInterchangeWarning(  
    BSTR* retval  
) ;
```

Return Value

A string describing the oldest interchangeability warning or empty string if no warnings remain.

IviDriverOperation.InterchangeCheck Property

-NOT SUPPORTED- If True, the driver maintains a record of interchangeability warnings. If the driver does not support interchangeability checking, attempts to set InterchangeCheck to True return an error.

Namespace: Ivi.Driver.Interop

Assembly: Ivi.Driver.Interop (in Ivi.Driver.Interop.DLL)

Syntax

Visual Basic

```
Public Property InterchangeCheck As Boolean
```

C#

```
public bool InterchangeCheck { get; set; }
```

Visual C++

```
HRESULT get_InterchangeCheck(  
    VARIANT_BOOL* val  
);  
HRESULT put_InterchangeCheck(  
    VARIANT_BOOL val  
);
```

IviDriverOperation.RecordCoercions Property

-NOT SUPPORTED- If True, the driver keeps a list of the value coercions it makes for VIInt32 and VIREal64 attributes. If the driver does not support coercion recording, attempts to set RecordCoercions to True will return an error.

Namespace: Ivi.Driver.Interop

Assembly: Ivi.Driver.Interop (in Ivi.Driver.Interop.DLL)

Syntax

Visual Basic

```
Public Property RecordCoercions As Boolean
```

C#

```
public bool RecordCoercions { get; set; }
```

Visual C++

```
HRESULT get_RecordCoercions(  
    VARIANT_BOOL* val  
);  
HRESULT put_RecordCoercions(  
    VARIANT_BOOL val  
);
```

IviDriverOperation.ResetInterchangeCheck Method

-NOT SUPPORTED- Resets the interchangeability checking algorithms of the driver so that methods and properties that executed prior to calling this function have no affect on whether future calls to the driver generate interchangeability warnings.

Namespace: Ivi.Driver.Interop

Assembly: Ivi.Driver.Interop (in Ivi.Driver.Interop.DLL)

Syntax

Visual Basic

```
Public Sub ResetInterchangeCheck ( _  
)
```

C#

```
public void ResetInterchangeCheck (  
)
```

Visual C++

```
HRESULT ResetInterchangeCheck (  
)?
```

IviDriverUtility.Disable Method

--NOT SUPPORTED-- Quickly places the instrument in a state where it has no, or minimal, effect on the external system to which it is connected. This state is not necessarily a known state.

Namespace: Ivi.Driver.Interop

Assembly: Ivi.Driver.Interop (in Ivi.Driver.Interop.DLL)

Syntax

Visual Basic

```
Public Sub Disable ( _  
)
```

C#

```
public void Disable(  
)
```

Visual C++

```
HRESULT Disable(  
)?
```

IviDriverUtility.ErrorQuery Method

--NOT SUPPORTED-- Queries the instrument and returns instrument specific error information. This function can be used when QueryInstrumentStatus is True to retrieve error details when the driver detects an instrument error.

Namespace: Ivi.Driver.Interop

Assembly: Ivi.Driver.Interop (in Ivi.Driver.Interop.DLL)

Syntax

Visual Basic

```
Public Sub ErrorQuery (
    ByRef ErrorCode As Integer
    ByRef ErrorMessage As String _
)
```

C#

```
public void ErrorQuery(
    ref int ErrorCode,
    ref string ErrorMessage
)
```

Visual C++

```
HRESULT ErrorQuery(
    long* ErrorCode,
    BSTR* ErrorMessage
);
```

Parameters

ErrorCode

Instrument error code

ErrorMessage

Instrument error message

IviDriverUtility.ResetWithDefaults Method

--NOT SUPPORTED-- Does the equivalent of Reset and then, (1) disables class extension capability groups, (2) sets attributes to initial values defined by class specs, and (3) configures the driver to option string settings used when Initialize was last executed.

Namespace: Ivi.Driver.Interop

Assembly: Ivi.Driver.Interop (in Ivi.Driver.Interop.DLL)

Syntax

Visual Basic

```
Public Sub ResetWithDefaults ( _  
)
```

C#

```
public void ResetWithDefaults (  
)
```

Visual C++

```
HRESULT ResetWithDefaults (  
) ;
```

Unsupported IVI-C APIs

The IVI-C APIs that are not supported are shown below.

KtM9188 IVI Driver Reference

Keysight Technologies

KTM9188_ATTR_INTERCHANGE_CHECK Attribute

-NOT SUPPORTED- If True, the driver maintains a record of interchangeability warnings. If the driver does not support interchangeability checking, attempts to set InterchangeCheck to True return an error.

Attribute Tree Node: \KtM9188\Inherent IVI Attributes\User Options\Interchange Check

Declaration: KtM9188.h

Implementation: KtM9188.dll

Syntax

```

Visual C++
#define KTM9188_ATTR_INTERCHANGE_CHECK 1050021
ViStatus KtM9188_GetAttributeViBoolean(
    ViSession Vi,
    ViConstString RepCapIdentifier,
    ViAttr AttributeID,
    ViBoolean* AttributeValue
);
ViStatus KtM9188_SetAttributeViBoolean(
    ViSession Vi,
    ViConstString RepCapIdentifier,
    ViAttr AttributeID,
    ViBoolean AttributeValue
);

```

Parameters

Vi

The ViSession handle that you obtain from the IviDriver_init or IviDriver_InitWithOptions function. The handle identifies a particular instrument session.

RepCapIdentifier

Must be VI_NULL or an empty string. This attribute is not defined on a repeated capability.

AttributeID

Must be KTM9188_ATTR_INTERCHANGE_CHECK.

AttributeValue (GetAttribute)

Returns the current value of the attribute. The user must specify the address of a variable that has the same data type as the attribute.

AttributeValue (SetAttribute)

The value to which to set the attribute.

KTM9188_ATTR_RECORD_COERCIONS Attribute

-NOT SUPPORTED- If True, the driver keeps a list of the value coercions it makes for ViInt32 and ViReal64 attributes. If the driver does not support coercion recording, attempts to set RecordCoercions to True will return an error.

Attribute Tree Node: \KTM9188\Inherent IVI Attributes\User Options\Record Value Coercions

Declaration: KtM9188.h

Implementation: KtM9188.dll

Syntax

```

Visual C++
#define KTM9188_ATTR_RECORD_COERCIONS 1050006

ViStatus KtM9188_GetAttributeViBoolean(
    ViSession Vi,
    ViConstString RepCapIdentifier,
    ViAttr AttributeID,
    ViBoolean* AttributeValue
);

ViStatus KtM9188_SetAttributeViBoolean(
    ViSession Vi,
    ViConstString RepCapIdentifier,
    ViAttr AttributeID,
    ViBoolean AttributeValue
);
    
```

Parameters

Vi

The ViSession handle that you obtain from the IviDriver_init or IviDriver_InitWithOptions function. The handle identifies a particular instrument session.

RepCapIdentifier

Must be VI_NULL or an empty string. This attribute is not defined on a repeated capability.

AttributeID

Must be KTM9188_ATTR_RECORD_COERCIONS.

AttributeValue (GetAttribute)

Returns the current value of the attribute. The user must specify the address of a variable that has the same data type as the attribute.

AttributeValue (SetAttribute)

The value to which to set the attribute.

KtM9188_ClearInterchangeWarnings Function

-NOT SUPPORTED- Clears the list of interchangeability warnings that the IVI specific driver maintains.

Function Tree Node: \KtM9188\Utility\CLEAR Interchange Warnings

Declaration: KtM9188.h

Implementation: KtM9188.dll

Syntax

Visual C++

```
ViStatus KtM9188_ClearInterchangeWarnings(  
    ViSession Vi  
);
```

Parameters

Vi

The ViSession handle that you obtain from the IviDriver_init or IviDriver_InitWithOptions function. The handle identifies a particular instrument session.

Return Value

Success or failure code.

KtM9188_Disable Function

--NOT SUPPORTED-- Quickly places the instrument in a state where it has no, or minimal, effect on the external system to which it is connected. This state is not necessarily a known state.

Function Tree Node: \KtM9188\Utility\Disable

Declaration: KtM9188.h

Implementation: KtM9188.dll

Syntax

Visual C++

```
ViStatus KtM9188_Disable(  
    ViSession Vi  
);
```

Parameters

Vi

The ViSession handle that you obtain from the IviDriver_init or IviDriver_InitWithOptions function. The handle identifies a particular instrument session.

Return Value

Success or failure code.

KtM9188_error_query Function

--NOT SUPPORTED-- Queries the instrument and returns instrument specific error information. This function can be used when QueryInstrumentStatus is True to retrieve error details when the driver detects an instrument error.

Function Tree Node: \KtM9188\Utility>Error Query

Declaration: KtM9188.h

Implementation: KtM9188.dll

Syntax

Visual C++

```
ViStatus KtM9188_error_query(  
    ViSession Vi,  
    ViInt32* ErrorCode,  
    ViChar[] ErrorMessage  
);
```

Parameters

Vi

The ViSession handle that you obtain from the IviDriver_init or IviDriver_InitWithOptions function. The handle identifies a particular instrument session.

ErrorCode

Instrument error code

ErrorMessage

Instrument error message

Return Value

Success or failure code.

KtM9188_GetNextCoercionRecord Function

-NOT SUPPORTED- Returns the oldest record from the coercion record list. Records are only added to the list if RecordCoercions is True.

Function Tree Node: \KtM9188\Utility\Get Next Coercion Record

Declaration: KtM9188.h

Implementation: KtM9188.dll

Syntax

Visual C++

```
ViStatus KtM9188_GetNextCoercionRecord(  
    ViSession Vi,  
    ViInt32 CoercionRecordBufferSize,  
    ViChar[] CoercionRecord  
);
```

Parameters

Vi

The ViSession handle that you obtain from the IviDriver_init or IviDriver_InitWithOptions function. The handle identifies a particular instrument session.

CoercionRecordBufferSize

The number of bytes in the ViChar array that the user specifies for the CoercionRecord parameter.

CoercionRecord

The coercion record string shall contain the following information: (1) The name of the attribute that was coerced. This can be the generic name, the COM property name, or the C defined constant. (2) If the attribute is channel-based, the name of the channel. The channel name can be the specific driver channel string or the virtual channel name that the user specified. (3) If the attribute applies to a repeated capability, the name of the repeated capability. The name can be the specific driver repeated capability token or the virtual repeated capability name that the user specified. (4) The value that the user specified for the attribute. (5) The value to which the attribute was coerced.

Return Value

Success or failure code.

KtM9188_GetNextInterchangeWarning Function

-NOT SUPPORTED- Returns the oldest warning from the interchange warning list. Records are only added to the list if InterchangeCheck is True.

Function Tree Node: \KtM9188\Utility\Get Next Interchange Warning

Declaration: KtM9188.h

Implementation: KtM9188.dll

Syntax

Visual C++

```
ViStatus KtM9188_GetNextInterchangeWarning(  
    ViSession Vi,  
    ViInt32 InterchangeWarningBufferSize,  
    ViChar[] InterchangeWarning  
);
```

Parameters

Vi

The ViSession handle that you obtain from the IviDriver_init or IviDriver_InitWithOptions function. The handle identifies a particular instrument session.

InterchangeWarningBufferSize

The number of bytes in the ViChar array that the user specifies for the InterchangeWarning parameter.

InterchangeWarning

A string describing the oldest interchangeability warning or empty string if no warnings remain.

Return Value

Success or failure code.

KtM9188_ResetInterchangeCheck Function

-NOT SUPPORTED- Resets the interchangeability checking algorithms of the driver so that methods and properties that executed prior to calling this function have no effect on whether future calls to the driver generate interchangeability warnings.

Function Tree Node: \KtM9188\Utility\Reset Interchange Check

Declaration: KtM9188.h

Implementation: KtM9188.dll

Syntax

Visual C++

```
ViStatus KtM9188_ResetInterchangeCheck (  
    ViSession Vi  
);
```

Parameters

Vi

The ViSession handle that you obtain from the IviDriver_init or IviDriver_InitWithOptions function. The handle identifies a particular instrument session.

Return Value

Success or failure code.

KtM9188_ResetWithDefaults Function

--NOT SUPPORTED-- Does the equivalent of Reset and then, (1) disables class extension capability groups, (2) sets attributes to initial values defined by class specs, and (3) configures the driver to option string settings used when Initialize was last executed.

Function Tree Node: \KtM9188\Utility\Reset With Defaults

Declaration: KtM9188.h

Implementation: KtM9188.dll

Syntax

Visual C++

```
ViStatus KtM9188_ResetWithDefaults(
    ViSession Vi
);
```

Parameters

Vi

The ViSession handle that you obtain from the IviDriver_init or IviDriver_InitWithOptions function. The handle identifies a particular instrument session.

Return Value

Success or failure code.

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 IS INTENTIONALLY LEFT BLANK.

Contact us

To obtain service, warranty, or technical assistance, contact us at the following phone or fax numbers:

United States:

(tel) 800 829 4444 (fax) 800 829 4433

Canada:

(tel) 877 894 4414 (fax) 800 746 4866

China:

(tel) 800 810 0189 (fax) 800 820 2816

Europe:

(tel) 31 20 547 2111

Japan:

(tel) (81) 426 56 7832 (fax) (81) 426 56
7840

Korea:

(tel) (080) 769 0800 (fax) (080) 769 0900

Latin America:

(tel) (305) 269 7500

Taiwan:

(tel) 0800 047 866 (fax) 0800 286 331

Other Asia Pacific Countries:

(tel) (65) 6375 8100 (fax) (65) 6755 0042

Or visit Keysight World Wide Web at:

www.keysight.com/find/assist

Product specifications and descriptions in this document are subject to change without notice. Always refer to Keysight Web site for the latest revision.

This information is subject to change without notice.
© Keysight Technologies 2014
Edition 1, August 2014



M9188-90006
www.keysight.com